

University of Dayton

eCommons

Graduate Theses and Dissertations

Theses and Dissertations

2005

Nonuniformity correction for focal plane arrays and partition based superresolution of video with application to an infrared imaging system

Balaji Narayanan
University of Dayton

Follow this and additional works at: https://ecommons.udayton.edu/graduate_theses

Recommended Citation

Narayanan, Balaji, "Nonuniformity correction for focal plane arrays and partition based superresolution of video with application to an infrared imaging system" (2005). *Graduate Theses and Dissertations*. 4627.
https://ecommons.udayton.edu/graduate_theses/4627

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, ecommons@udayton.edu.

NONUNIFORMITY CORRECTION FOR FOCAL PLANE ARRAYS
AND PARTITION BASED SUPERRESOLUTION OF VIDEO
WITH APPLICATION TO AN INFRARED
IMAGING SYSTEM

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

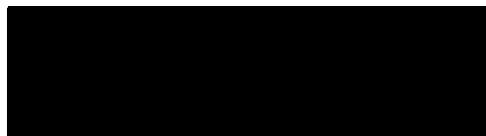
by

Balaji Narayanan
UNIVERSITY OF DAYTON
Dayton, Ohio

December 2005

NONUNIFORMITY CORRECTION FOR FOCAL PLANE ARRAYS
AND PARTITION BASED SUPERRESOLUTION OF VIDEO WITH
APPLICATION TO AN INFRARED IMAGING SYSTEM

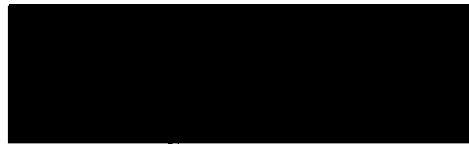
APPROVED BY:



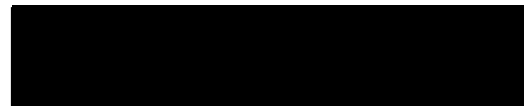
Russell C. Hardie Ph.D.
Advisory Committee Chairman
Associate Professor, Electrical and
Computer Engineering Department



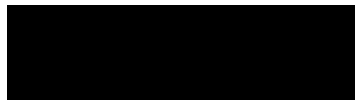
John S. Loomis Ph.D.
Committee Member
Associate Professor, Electrical and
Computer Engineering Department



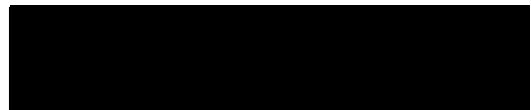
Raul Ordonez Ph.D.
Committee Member
Associate Professor, Electrical and
Computer Engineering Department



Kenneth J. Barnard Ph.D.
Committee Member
Research Engineer, Air Force
Research Laboratory



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



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

TABLE OF CONTENTS

	Page
List of Figures	vi
List of Tables	x
Abstract	xi
Dedication	xiii
Acknowledgments	xiv
Chapters:	
1. INTRODUCTION	1
1.1 The Problem	1
1.1.1 Spatial Nonuniformity	1
1.1.2 Loss of Spatial Resolution	2
1.2 The Contribution	5
1.3 Prior Work	6
1.3.1 Overview of NUC Algorithms	6
1.3.2 Overview of SR Algorithms	7
1.4 Organization of Dissertation	8
2. SPATIAL NONUNIFORMITY MODEL	10
2.1 Detector Level Model	10
2.2 Readout Amplifier Model	11

3.	SCENE BASED NUC ALGORITHM DEVELOPMENT	14
3.1	Readout Amplifier NUC	14
3.1.1	Local Constant Statistics Constraint	15
3.1.2	Implementation Issues	17
3.2	Detector Level NUC	17
3.2.1	Initial Scene Estimate Using Median Filtering	18
3.2.2	Recursive Least Squares NUC Parameter Estimation	18
3.3	Final Nonuniformity Correction	20
4.	PERFORMANCE ANALYSIS OF NUC ALGORITHM	21
4.1	Simulated Nonuniformity	21
4.2	Real Infrared Data	27
4.2.1	Amber Infrared Imagery	27
4.2.2	Night Conqueror Infrared Imagery	30
5.	REGION-BASED WNN NONUNIFORM INTERPOLATION	33
5.1	WNN SR Approach	34
5.1.1	Wiener Filtering	35
5.2	Modified WNN HR Image Reconstruction	36
5.2.1	Results	39
6.	PARTITION-BASED SUPERRESOLUTION FOR VIDEO PROCESSING	44
6.1	Partition Weighted Sum Filters	44
6.2	Super-resolution Observation Model	45
6.3	Proposed Super-resolution Approach	47
6.3.1	Algorithm Overview	47
6.3.2	Super-resolution Filters	49
7.	OPTIMIZATION AND COMPUTATIONAL COMPLEXITY OF PWS	
	SR filters	53
7.1	Filter Optimization	53
7.2	Computational Complexity	55
7.2.1	Translational Motion	56
7.2.2	Arbitrary Motion	60

8.	HR VIDEO RECONSTRUCTION AND RESTORATION USING PWS SR FILTERS	62
8.1	Simulated Data	62
8.2	Infrared Data	69
8.2.1	Translational Motion	72
8.2.2	Translation and Rotation	77
9.	CONCLUSIONS	82
Appendices:		
A.	Matlab Source Code For NUC Algorithm	84
B.	Matlab Source Code For Region-Based WNN Nonuniform Interpolation .	98
C.	Matlab Source Code For PWS-Based SR of Video	117
	Bibliography	136

LIST OF FIGURES

Figure	Page
2.1 Channel level readout pattern for (a) Night conqueror II infrared camera (b) Amber infrared camera.	12
3.1 Block diagram of the proposed NUC algorithm.	15
4.1 True visible 8-bit image.	22
4.2 Frame 100 in simulated image sequence (a) Uncorrupted true frame, (b) Corrupted with simulated channel and detector nonuniformity, (c) Corrected for readout nonuniformity, (d) Preliminary scene estimate using 5×5 median filter, (e) Corrected using RLS algorithm with 100 frames, (f) Corrected using RLS algorithm without applying readout correction.	24
4.3 Error metrics to evaluate the performance of the proposed algorithm as a function of median filter size. (a) MAE, (b) SNR.	26
4.4 Frame 100 in real infrared image sequence acquired using Amber infrared imager. (a) Raw frame, (b) Corrected for readout nonuniformity, (c) Preliminary scene estimate using 3×3 median filter, (d) Corrected using RLS algorithm with 100 frames.	28
4.5 Frame 100 of Amber infrared image sequence corrected using RLS algorithm without applying readout correction.	29
4.6 Frame 100 in real infrared image sequence acquired using Night conqueror infrared imager. (a) Raw frame, (b) Corrected for readout nonuniformity, (c) Preliminary scene estimate using 13×13 median filter, (d) Corrected using RLS algorithm with 100 frames.	31

5.1	HR grid with nonuniformly spaced LR samples.	34
5.2	HR image reconstructed using WNN SR algorithm	37
5.3	Flowchart for modified WNN SR algorithm.	38
5.4	(a) HR image reconstructed in the region of overlap on the HR grid (b) new region reconstructed after eliminating frame 4 that contributes the least area of overlap with the HR grid.	38
5.5	Frame 1 from the low resolution infrared sequence.	39
5.6	Horizontal and vertical shifts in pixels for each low resolution frame. .	40
5.7	HR image estimate using WNN SR algorithm after Wiener filtering. .	41
5.8	HR image estimate using modified WNN SR algorithm with maximum number of neighbor=4 and minimum number of neighbor=4 (a) before Wiener filtering (b) after Wiener filtering.	42
5.9	(a) Region reconstructed using modified WNN SR algorithm with all the 16 LR frames (b) Region reconstructed using modified WNN SR algorithm with minimum number of neighbor=1.	42
6.1	Block diagram of PWS filter	45
6.2	Observation model for image superresolution.	46
6.3	Overview of the proposed SR algorithm.	47
6.4	Example of a partially populated HR grid. Missing pixels are shown in black.	48
6.5	$N_1 \times N_2$ local observation window.	49
6.6	Number of PWS SR filters required for $M = 10$ partitions.	52
7.1	Block diagram illustrating the filter optimization procedure.	54
7.2	$N_1 \times N_2$ moving observation window on HR grid for (a) $L_1 = L_2 = 3$ and $K_1 = K_2 = 5$ (b) $K_1 = L_1 = K_2 = L_2 = 3$	57

7.3	Flops per output HR pixel as a function of the HR grid density fraction f for translational motion with $N = 15 \times 15$, $S_1 = S_2 = 64$, and $M = 1$.	59
7.4	Flops per output HR pixel as a function of the HR grid density fraction f for arbitrary motion compared with the computational complexity of translational motion with $N = 15 \times 15$, $S_1 = S_2 = 64$, $K_1 = K_2 = 5$, $L_1 = L_2 = 5$ and $M = 1$.	60
8.1	Simulation images. (a) Original 8-bit image, (b) simulated noisy low resolution frame.	63
8.2	Simulation parameters. (a) Position of the simulated LR frames relative to the original HR image, (b) combination of simulated LR frames used to generate the different length input sequences.	64
8.3	8-bit training image used to obtain VQ codebook, and the PWS autocorrelation and crosscorrelation matrices for the simulation results and for the infrared image results.	64
8.4	VQ codebook for $M = 10$ partitions.	65
8.5	Autocorrelation matrices for the different partitions. (a) $M = 1$, (b) $M = 2$, (c) $M = 3$, (d) $M = 4$, (e) $M = 5$, (f) $M = 6$, (g) $M = 7$, (h) $M = 8$, (i) $M = 9$, (j) $M = 10$.	66
8.6	Error for PWS SR filter as function of number of partitions (a) MAE (b) MSE.	67
8.7	Mean absolute error (MAE) as a function of the number of input frames, Q , for various SR methods applied to the simulated image sequences.	68
8.8	HR image estimate for several SR algorithms using $Q = 5$ frames. (a) PWS SR filter with $M = 1$, $N_1 = N_2 = 15$, and $K_1 = K_2 = 5$. (b) PWS SR filters with $M = 10$, $N_1 = N_2 = 15$, $P_1 = P_2 = 7$, and $K_1 = K_2 = 5$. (c) RLS image reconstruction technique (d) WNN method (e) Delaunay triangulation (f) bicubic interpolation of Frame 1.	71
8.9	LR frame 1 of infrared image sequence data with translational motion.	73

8.10	Registration parameters in HR pixels for the $Q = 15$ frame infrared image sequence with translational motion.	73
8.11	SR images generated from $Q = 15$ infrared image frames with translational motion. (a) PWS SR filters with $M = 1$, $N_1 = N_2 = 15$ and $K_1 = K_2 = 5$, (b) PWS SR filters with $M = 10$, $N_1 = N_2 = 15$, $P_1 = P_2 = 7$ and $K_1 = K_2 = 5$, (c) RLS image reconstruction algorithm, (d) WNN method, (e) Delaunay triangulation, (f) single frame bicubic interpolation.	75
8.12	Full size HR image after border processing using PWS SR filters. . .	76
8.13	LR frame 1 of infrared image sequence data with translational and rotational motion.	78
8.14	Registration parameters in HR pixels for the $Q = 20$ frame infrared image sequence with translational and rotational motion.	78
8.15	SR images generated from $Q = 20$ infrared image frames with translational and rotational motion. (a) PWS SR filter with $M = 1$, $N_1 = N_2 = 15$ and $K_1 = K_2 = 5$ (b) PWS SR filter with $M = 10$, $N_1 = N_2 = 15$, $P_1 = P_2 = 7$ and $K_1 = K_2 = 5$ (c) RLS image reconstruction algorithm (d) bicubic interpolation of frame 1.	81

LIST OF TABLES

Table		Page
3.1	Recursive least squares nonuniformity parameter estimation, performed for each pixel j	19
4.1	Mean and standard deviation (std) for the gain and the bias for each of the four channels to simulate the channel nonuniformity.	22
8.1	Run time and flops per output HR pixel for FLIR image results with translational motion.	77
8.2	Run time and flops per output HR pixel for FLIR results with translational and rotational motion.	79

ABSTRACT

NONUNIFORMITY CORRECTION FOR FOCAL PLANE ARRAYS AND PARTITION BASED SUPERRESOLUTION OF VIDEO WITH APPLICATION TO AN INFRARED IMAGING SYSTEM

Name: Narayanan, Balaji
University of Dayton, December 2005

Advisor: Dr Russell C. Hardie

The utility of many infrared imaging systems is known to be affected by two common problems. The images acquired using an infrared imaging system is corrupted by the spatial fixed pattern noise. Spatial fixed pattern noise is caused by the variation in the photoresponses of the individual detectors in the focal plane array and by the nonuniform response of the readout and digitization electronics involved in multiplexing the individual detector responses. In addition, the spatial nonuniformity tends to drift in time due to the variation in the surrounding conditions.

The second problem associated with any imaging system is that the detectors placed at the focal plane array of the imaging system are not close enough to sufficiently sample the scene at the Nyquist rate. The acquired images are degraded by the aliasing effects. The spatial frequencies could be sufficiently bandlimited for the chosen detector array by reducing the aperture of the optics. However, some aliasing is often preferred to a loss of optical resolution in the design of imaging systems.

In this dissertation, we describe a new scene based nonuniformity correction (NUC) algorithm that treats the aggregate spatial nonuniformity in separate stages. One advantage of the proposed algorithm is that it requires only few frames to obtain high quality corrections. The effectiveness of the proposed approach is tested by applying it to simulated and infrared imagery.

We also investigate the use of partition-based weighted sum (PWS) filters for super-resolution (SR) of video. Though, the PWS filters have been previously used for image super-resolution, they are not suitable for video processing. Moreover, some of the existing SR algorithms are computationally intensive and may be difficult to implement in real time video system. In this dissertation, we present a novel approach for applying the partition-based SR filters to video that is computationally efficient and suitable for parallel implementation. The previously proposed PWS filters were used for image SR with a translation motion model. Here, a new implementation of the PWS filters for a general motion model is proposed. We also present a detailed computational analysis, quantitative comparisons with several benchmark techniques, and apply and evaluate the method with rotational motion as well as translational motion.

Dedicated to my loving parents, beloved wife Geetha, my brother Selva and friend
Bro. Kenneth Thompson.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Dr. Russell Hardie for all his expert guidance and encouragement. He is an excellent and outstanding professor. He has been a great source of inspiration for me through out my doctoral study at the University of Dayton. I greatly appreciate the financial support he has given me all through out my study. He has been very understanding and supportive at times of need.

I would like to thank Dr. John Loomis for taking time to read my dissertation and sit on the dissertation committee. He is an amazing professor from whom I have learned a great deal. I also thank him for his invaluable assistance he has provided me in regards to my research.

I would like to thank Dr. Raul Ordonez for taking time to read this dissertation and sit on my committee. I also thank them for his useful guidance in regards to my research.

I would like to thank Dr. Ken Barnard from the Air Force Research Laboratory for his willingness to serve as a committee member and for taking time to read my dissertation, amidst his busy schedule. I learnt a great deal from his random process course.

I would also like to thank both Lorreta Christon and Marilyn Knisley for all their help over the past few years. Further, I would like to thank Karen Hardie for taking

care of my contract and helping me through out my graduate study at the University of Dayton.

I would like to gratefully acknowledge CMC Electronics Cincinnati for providing the Night conqueror infrared imagery used for testing. In particular, I would like to thank Mr. Doug Droege for collecting, formatting and delivering image data. I would also like to thank Brian Yasuda of the FLIR research group in the Sensors Technology Branch, Air Force Research Laboratory, WPAFB, OH for providing the Amber infrared imagery.

I would like to thank my friend Mahendran Madhavan and Bro. Kenneth Thompson for their encouragement and support during my graduate study.

I would like to thank my parents for their love and support. They have been a great source of inspiration and encouragement through out my life. Finally, I would like to thank my amazing wife Geetha for all the tremendous support and encouragement she has given through out my life. Without her it would not have been an easy journey for me.

CHAPTER 1

INTRODUCTION

In this chapter, we describe the two important degradations involved during image acquisition using an infrared imaging system. The remainder of this chapter is organized as follows. The problems associated with an infrared imaging system is described in Section 1.1. Section 1.2 presents the novel contributions in this dissertation. Finally, a brief literature overview of the previous work is discussed in Section 1.3.

1.1 The Problem

1.1.1 Spatial Nonuniformity

Infrared imaging systems typically consist of an array of photodetectors at the focal plane of the imaging system. The images acquired using an infrared imaging system are degraded by the presence of the spatial fixed pattern noise. The spatial noise is due to the nonuniformity in the photoresponse of the individual detectors for the same irradiance. The performance of the infrared imaging system is further affected by the nonuniformity in the readout and the digitization electronics involved in multiplexing the individual detector responses. The spatial nonuniformity tends

to drift in time as the ambient conditions vary. This requires a correction technique that adapts to the changing detector responses.

There are two different types of NUC techniques. The standard approach normalizes the photoresponses using uniform calibration sources. For example, given two uniform sources, a linear correction can be applied to each detector (i.e., the so-called two-point correction method). The second approach is referred to as scene-based NUC. Here the images collected during the normal operation of the camera are used for NUC. The disadvantage of the two-point calibration method is that it requires the normal imaging operations be halted during correction. Furthermore, it requires calibration targets and a mechanical system to switch from imaging the scene to the calibration targets. Scene based NUC methods do not require the imaging operations to be halted while performing the nonuniformity correction; moreover they are capable of adapting to the changing detector responses arising due to variations in the operating conditions. As a result, scene-based NUC is a critical area of research.

1.1.2 Loss of Spatial Resolution

There are many tradeoffs to be considered when designing an imaging system for image acquisition. The desire for high spatial resolution and a wide field of view generally leads to camera system employing small $f/\#$ (F-number) optics. This produces an image with very high spatial bandwidth at the focal plane. A focal plane array (FPA) with sufficiently small detector spacing is required to sample the scene at the Nyquist rate [1]. However, several factors may make it impractical to equip the camera with an FPA with sufficiently small detector spacing to meet the Nyquist

criterion. Fabrication complexities and corresponding cost may be the limiting factor. More fundamentally, as detector sizes decrease, so does the amount of light seen by each detector, leading to potentially severe shot noise [2]. The spatial frequencies could be sufficiently bandlimited for the chosen detector array by reducing the aperture of the optics. However, some aliasing is often preferred to a loss of optical resolution in the design of imaging systems. Furthermore, the loss of optical resolution is far harder (if not impossible) to compensate for in post processing than undersampling. For these reasons, many imaging systems produce images that suffer from aliasing artifacts resulting from undersampling. Such a system may be considered to be detector limited with regards to spatial resolution. A very active area of research, often referred to as super-resolution, involves post-processing techniques to overcome such limitations [2].

In general, the concept of SR relates to a process whereby images are obtained with resolutions that are, in some way, beyond the limiting factors of the uncompensated imaging system. It is important to understand what the limiting factors are in an imaging system and to use the appropriate tool when seeking SR enhancement. To this end, we find it instructive to separate the concept of SR into two categories: optical SR, and detector SR. The goal of optical SR is to create an image with *valid* spatial frequency content that goes beyond that of the cut-off frequency of the optics [3]. Such techniques generally must rely on rich and accurate *a priori* information. On the other hand, detector SR seeks to overcome the limitations of the detector array and achieve the full resolution afforded by the selected optics. This is done by essentially “unscrambling” the aliased frequency content present in the observed frames.

Detector SR algorithms generally process a set of low-resolution (LR) aliased frames from a video sequence to produce a high-resolution (HR) frame. When sub-pixel relative motion is present between the objects in the scene and the detector array, a unique set of scene sample is acquired with each frame. This provides the mechanism for effectively increasing the spatial sampling rate of the imaging system. The key to all of these techniques is accurate subpixel motion estimates.

In this dissertation, we focus on detector SR of video using PWS filters. PWS filters have been used for image restoration [4–6] and more recently Shao *et al.* [7] applied PWS filters to the problem of SR enhancement of still images. In the developed approach the PWS filters perform nonuniform interpolation and image restoration simultaneously. As with the other interpolation-restoration methods, the LR frames are registered and the motion estimates are used to partially populate an HR grid with the LR samples. The PWS filters operate with a moving window on the HR grid. At each window location, the output is formed using a weighted sum of the pixels present within the window. The weights used depend on the configuration of missing pixels in the window and the intensity structure of the present pixels. The intensity structure is classified using vector quantization (VQ). In [7], the entire training procedure for determining the filter weights is carried out after the registration step, which determines the specific locations on the HR grid that are populated. This technique is not suitable for resolution enhancement of video because the configuration of the HR grid changes when different LR frames are observed during video processing. Furthermore, only global translational motion was considered in [7].

1.2 The Contribution

In this dissertation, we present a scene based NUC technique that treats the nonuniformity in the observed frames in separate steps. The novelty in this approach is the modification and integration of the existing NUC algorithms described in work by Muse [8] and Hardie [9]. In the first step, the nonuniformity due to the readout electronics is corrected by grouping the pixels that belong to the same readout amplifier channel. We normalize these channel outputs so that each channel produces pixels with the same mean and standard deviation as pixels from the other channels [8]. This is followed by the nonuniformity correction at the individual detector level using nonlinear filter based-approach [9]. A spatial median filter is used to obtain a preliminary estimate of the true scene from the frames corrected for readout nonuniformity. The individual detector gain and bias parameters are estimated using recursive least squares (RLS) curve fitting between the preliminary scene estimate and the corresponding frame corrected for readout nonuniformity for each pixel. The performance of the proposed algorithm is demonstrated by applying it to simulated imagery and real infrared data. Quantitative results in terms of mean absolute error and signal to noise ratio are also presented to demonstrate the efficacy of the proposed algorithm.

The main contribution of this dissertation is primarily with the innovative PWS filter training and implementation methodology for computationally efficient SR of video. As in [7] the partially populated HR grid is generated and PWS filters are employed to simultaneously perform nonuniform interpolation (to fully populate the high resolution grid) and perform deconvolution of the system point spread function.

The filter weights are selected from a filter bank based on the configuration of missing pixels in the window and the intensity structure of the present pixels. With the proposed algorithm, the bulk of the filter training is done off-line. The new implementation of the PWS filters for SR can be applied to a general motion model. We believe that, in addition to SR image enhancement, the proposed technique is a very useful tool for addressing the general problem of nonuniform interpolation. For the case of translational motion, minimal computations are required per frame to obtain all the needed filter weights. A number of experimental results are presented to demonstrate the efficacy of the proposed algorithm in comparison to several previously published methods. In our experimental results, the proposed method generally outperformed all of the benchmark methods and has one of the lowest computational complexities.

1.3 Prior Work

1.3.1 Overview of NUC Algorithms

There are a wide variety of scene-based NUC algorithms proposed in the literature. Narendra *et al.* [10], Harris *et al.* [11, 12] and Chiang *et al.* [13], developed scene-based NUC algorithms based on the assumption that the temporal means and variances are identical for all pixels. O’Neil *et al.* [14] and Hardie *et al.* [15], developed NUC algorithms based on the fact that detectors that observe the same scene point over different times should have the same response. Ratliff *et al.* [16] developed an algebraic scene-based NUC technique that corrects only the offset variations in the detector using a gradient-based shift estimator. Hayat *et al.* [17] developed a statistical algorithm based on the constant-range assumption. Torres *et al.* [18] developed an adaptive scene-based NUC algorithm using a neural network approach.

1.3.2 Overview of SR Algorithms

An excellent survey of detector SR techniques can be found in [2]. Detector SR techniques may be grouped into three categories: frequency domain approaches, iterative HR image reconstruction techniques, and interpolation-restoration approaches. Let us briefly review prior work in each of these areas. Tsai and Hunag [19] proposed a frequency domain detector SR algorithm. They exploit the shift property of the Fourier transform and develop a system of equations in the frequency domain that relates the high resolution image to the observed LR frames. Kim *et al.* [20, 21] proposed a recursive least squares solution by extending the approach in [19] to include the effects of motion blur and additive noise. The Fourier domain based techniques are limited to global translation models and cannot be used for general motion models that include spatial variations.

A variety of iterative HR image reconstruction algorithms have been proposed. Basically, these seek to create an HR image estimate that, when put through the image observation model, produces simulated LR frames that closely match the observed frames and is consistent with certain *a priori* assumptions. Iterated back projection methods have been employed in [1, 22–28]. Stochastic methods derived using a Bayesian framework can be found in [29–32]. All of these iterative methods generally perform well. However, due to their iterative nature, they may be difficult to implement in a real-time video system.

The interpolation-restoration type methods are perhaps the most straightforward and computationally simple methods. Here the LR pixels are used to partially populate a HR grid based on the subpixel-motion estimates. In general, these LR pixels will fall nonuniformly on the HR grid. Nonuniform interpolation is used to obtain

values on a uniform HR grid. This creates a more finely sampled scene. The image at this point appears to be one acquired with a detector array that has reduced pixel spacing, but the size of each detector remains the same. Thus, an image restoration technique is applied to deconvolve the detector blur and possibly the diffraction blur from the optics as well. Interpolation-restoration approaches for detector SR are proposed in [7, 33–42]. In [39] a wavelet based interpolation followed by regularized restoration is employed. A weighted nearest-neighbor (WNN) nonuniform interpolation technique is proposed in [36, 40] to reconstruct an HR image from a sequence of LR frames with pure translational motion. Wiener filtering is used to restore the image from noise and the blurring effects introduced by the detectors and optics. Lertrattanapanich *et al.* [42] proposed an interpolation technique based on Delaunay triangulation for improving the spatial resolution. A related problem of estimating missing data in images is addressed in [43].

1.4 Organization of Dissertation

The organization of this dissertation is as follows. In Chapter 2, the observation model for spatial nonuniformity is developed. In particular, the individual detector level model and readout architecture based model for nonuniformity is defined. The proposed NUC algorithm is discussed in Chapter 3. The performance of the proposed NUC algorithm is analyzed in Chapter 4. The experimental results obtained by applying the proposed NUC algorithm to simulated imagery and true infrared imagery is also presented. In Chapter, 5 the WNN SR approach is briefly described. A new region-based nonuniform interpolation approach to reconstruct the borders of the HR image is proposed. Experimental results obtained using the region-based

approach is also included. The operation of PWS filters is discussed in Chapter 6. The observation model for SR and the proposed PWS SR filters for SR video are also defined. Chapter 7 contains a description of the new training scheme for PWS SR filters. The optimization and computational efficiency of the PWS SR filters is discussed. The application of the proposed PWS SR filters to simulated and true infrared video sequence is presented in Chapter 8. Quantitative analysis of the PWS SR filters is also performed. Finally, Chapter 9 consists of few concluding remarks and possible future work. The Matlab source code to generate the figures in Chapters 4, 5 and 8 are included in Appendices A, B and C, respectively.

CHAPTER 2

SPATIAL NONUNIFORMITY MODEL

We begin this chapter by introducing the observation model developed for the spatial nonuniformity. Most NUC algorithms model the aggregate detector and readout nonuniformity using single gain and bias. We model the detector and readout nonuniformity separately and correct them individually. The remainder of the chapter is organized as follows: Section 2.1 presents the detector level model and the readout amplifier model is introduced in Section 2.2. The notations and definitions introduced in this chapter are meaningful till the end of Chapter 4.

2.1 Detector Level Model

Consider an infrared imager consisting of P infrared detectors in the focal plane array of the imaging system. Let K be the number of frames generated by the infrared imager. The true infrared radiation incident on detector j in frame k be denoted $x_k(j)$, where $1 \leq j \leq P$ and $1 \leq k \leq K$. For $N \geq 1$, the output pixel value of this detector is modelled as an N 'th order polynomial given by

$$\begin{aligned} y_k(j) &= a_{k,1}(j)x_k^N(j) + a_{k,2}(j)x_k^{N-1}(j) + \dots \\ &\quad + a_{k,N+1}(j) = \mathbf{a}_k(j)^T \mathbf{x}_k(j), \end{aligned} \tag{2.1}$$

where

$$\mathbf{a}_k(j) = [a_{k,1}(j), a_{k,2}(j), \dots, a_{k,N+1}(j)]^T \quad (2.2)$$

are the nonuniformity parameters for detector j in frame k , and

$$\mathbf{x}_k(j) = [x_k^N(j), x_k^{N-1}(j), \dots, x_k(j), 1]. \quad (2.3)$$

This model is equivalent to the standard gain and bias model for $N = 1$ and includes nonlinear responses for $N > 1$.

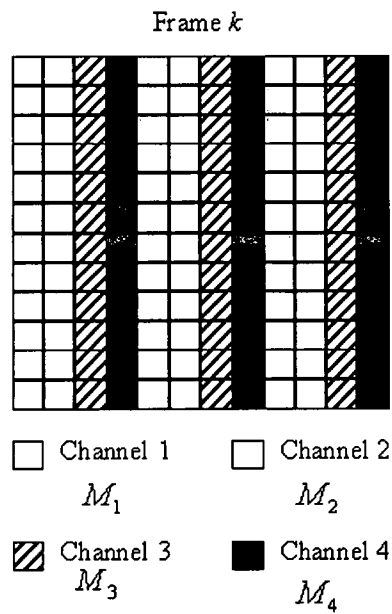
2.2 Readout Amplifier Model

Focal plane arrays (FPA) generally have multiple output channels in order to meet the high-bandwidth requirement imposed by the infrared cameras. The photoresponse of the detectors are grouped and multiplexed into each output channel using different readout amplifier circuits. Each group of photodiode signals is processed by an amplifier circuit characterized by distinct gain and bias parameters. This induces a uniform gain and bias error in the photoresponse of the detectors belonging to each group. The individual detector outputs given by (2.1) for $1 \leq j \leq P$ and $1 \leq k \leq K$ are grouped and multiplexed using a readout amplifier into each output channel. In particular, consider L groups of outputs, where the elements of each group come from a common readout amplifier. Let us define sets of detector indices corresponding to each group as M_1, M_2, \dots, M_L . The set of detector indices are mutually exclusive such that

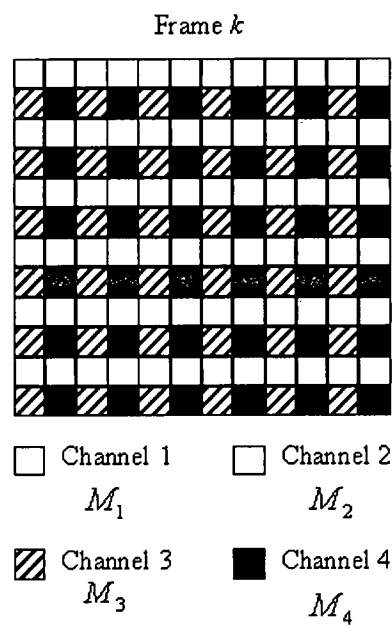
$$M_1 \cup M_2 \dots \cup M_L = \{1, 2, \dots, P\} \quad (2.4)$$

and

$$M_i \cap M_j = \emptyset \quad (2.5)$$



(a)



(b)

Figure 2.1: Channel level readout pattern for (a) Night conqueror II infrared camera
(b) Amber infrared camera.

for $i \neq j$. The number of output pixels in group i in each frame be denoted P_i for $i = 1, 2, \dots, L$. The total number of pixels in a frame is given by

$$P = \sum_{i=1}^L P_i. \quad (2.6)$$

Figures 2.1(a) and 2.1(b) show an example of the channel amplifier output configuration for the Night Conqueror II FPA and the Amber FPA respectively. The Night Conqueror II FPA is configured in such a way that pixels belonging to each column of a frame are multiplexed onto one of four output channels, for the Amber FPA pixels from alternative column and row of a frame are multiplexed onto one of four output channels. Here, each group amplifier is characterized by a linear model. The input-output relationship for each group for frames 1 through k can be expressed as

$$z_k(j) = b_{k,i}y_k(j) + c_{k,i}, \quad (2.7)$$

for $j \in M_i$, $i = 1, 2, \dots, L$ and $k = 1, 2, \dots, K$, where $b_{k,i}$ and $c_{k,i}$ are the gain and bias parameters for the i 'th group amplifier for frame k , respectively.

CHAPTER 3

SCENE BASED NUC ALGORITHM DEVELOPMENT

In this chapter, we present the NUC algorithm developed to treat the spatial nonuniformity. The proposed algorithm is divided into two stages. The first stage corrects for the nonuniformity arising from the nonuniform behavior of the readout amplifiers. The frames corrected for the readout nonuniformity are then corrected to treat for the nonuniformity resulting from the variation in the individual detector responses. The remainder of this chapter is organized as follows: Section 3.1 describes the readout architecture based NUC and the detector level NUC is presented in Section 3.2.

3.1 Readout Amplifier NUC

The different stages involved in the proposed NUC algorithm are illustrated as shown in Fig. 3.1. The proposed algorithm begins by correcting the nonuniformity resulting from the readout channel amplifier. This is achieved by exploiting the knowledge of the FPA architecture for a given imaging system [8]. Based upon the readout architecture of the imaging system, we group the outputs of each channel and force them to have the same first and second order statistics. This is what we refer to as Local Constant Statistics (LCS) constraint described in the Section 3.1.1. In the

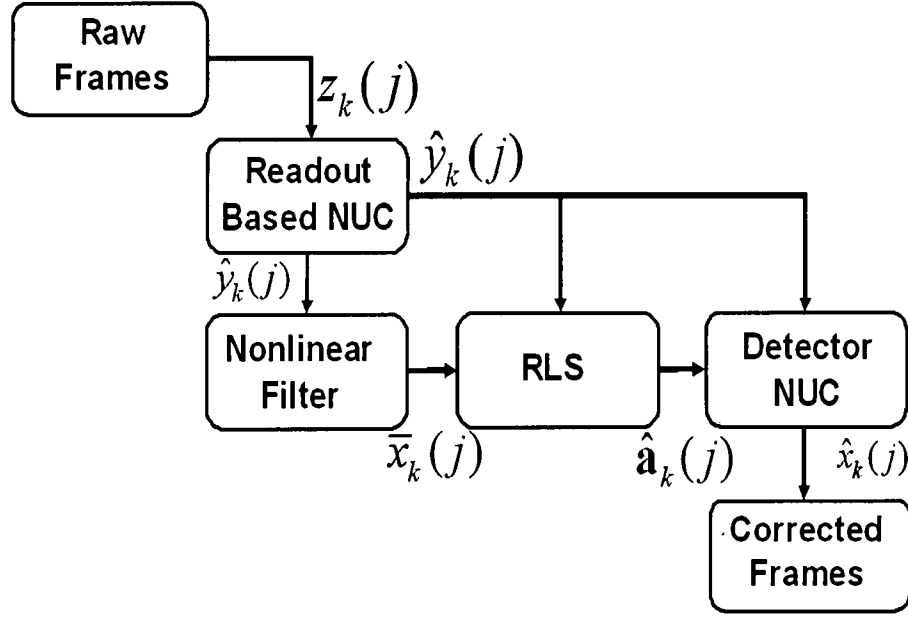


Figure 3.1: Block diagram of the proposed NUC algorithm.

case of the Night Conqueror II infrared camera and the Amber infrared camera, the pixels in each frame are split into four groups, one for each output channel as shown in Fig. 2.1(a) and 2.1(b) respectively. Since the pixels belonging to these groups are equally distributed about the image, the statistics for these groups should be similar. The groups formed from a single frame contain sufficient number of pixels to evaluate the statistics of each group. Thus, the channel level correction can be performed with as little as one frame.

3.1.1 Local Constant Statistics Constraint

The objective in the LCS constraint is to make the first and the second order statistics of the pixels belonging to group i similar to those from the groups in close

spatial proximity to group i . In order to realize this constraint, one needs to assume that the inputs of the groups neighboring group i , have the same first and second order statistics and the gain and biases are uniform across any sub-groups. Once the constraint is defined, it is possible to determine the gain and bias of each group amplifier from the pixel values of the observed frames. The channel level correction can be performed with as little as one frame because the groups are spatially well distributed and are populated with sufficient number of pixels.

The gain and the bias parameters for each group amplifier is determined by applying the LCS constraint and computing the sample mean and the standard deviation estimates of each group from the observed frames. The mean and standard deviation estimates for each group for frame k using the last R frames are given by

$$\mu_{k,i} = \frac{1}{RP_i} \sum_{n=\max(1,r)}^k \sum_{j \in M_i} z_n(j), \quad (3.1)$$

and

$$\sigma_{k,i} = \frac{1}{RP_i} \sum_{n=\max(1,r)}^k \sum_{j \in M_i} (z_n(j) - \mu_{k,i})^2, \quad (3.2)$$

respectively, for $r = k - R + 1$, $i = 1, 2, \dots, L$ and $k = 1, 2, \dots, K$. Based on the readout architecture, the Q_i groups neighboring group i are identified and the set of indices of these groups is denoted W_i . The averages of the first and second order statistics for the neighboring groups are formed as

$$\bar{\mu}_{k,i} = \frac{1}{Q_i} \sum_{n \in W_i} \mu_{k,n} \quad (3.3)$$

and

$$\bar{\sigma}_{k,i} = \frac{1}{Q_i} \sum_{n \in W_i} \sigma_{k,n}, \quad (3.4)$$

respectively. In order for each group to have the first and second order statistics equal to the average of the neighboring groups, we apply the following correction to obtain

the estimate of $y_k(j)$:

$$\hat{y}_k(j) = \left(\frac{z_k(j) - \mu_{k,i}}{\sigma_{k,i}} \right) \bar{\sigma}_{k,i} + \bar{\mu}_{k,i} \quad (3.5)$$

for $j \in M_i$ and $i = 1, 2, \dots, L$. The effective gain and the bias estimates from (2.7) are given by

$$\hat{b}_{k,i} = \frac{\sigma_{k,i}}{\bar{\sigma}_{k,i}} \quad (3.6)$$

and

$$\hat{c}_{k,i} = \mu_{k,i} - \frac{\sigma_{k,i}}{\bar{\sigma}_{k,i}} \bar{\mu}_{k,i}, \quad (3.7)$$

respectively.

3.1.2 Implementation Issues

The important part of the readout based NUC involves estimating the group mean and standard deviations. The total pixel count in each group increases with every new observed frame. This may make the direct computation of the group mean and standard deviation difficult and also demanding in terms of memory resources. The group mean and standard deviation estimates are computed recursively to simplify the process of implementation. The estimates computed recursively are given by

$$\hat{\mu}_{k,i} = \frac{k-1}{k} \hat{\mu}_{k-1,i} + \frac{1}{kP_i} \sum_{j \in M_i} z_k(j) \quad (3.8)$$

and

$$\hat{\sigma}_{k,i}^2 = \frac{k-1}{k} \hat{\sigma}_{k-1,i}^2 + \frac{1}{kP_i} \sum_{j \in M_i} (z_k(j) - \hat{\mu}_{k,i})^2. \quad (3.9)$$

3.2 Detector Level NUC

The second phase of the proposed algorithm following the readout based NUC is the individual detector level NUC. This is done using a nonlinear filter based

approach[9]. A median filter is used to estimate the true irradiance from the frames corrected for readout nonuniformity and a least squares curve fitting is used to identify the detector nonuniformity parameters based on the preliminary irradiance estimate.

3.2.1 Initial Scene Estimate Using Median Filtering

The first step in treating the individual detector level nonuniformity is to obtain a preliminary estimate of the true irradiance. The choice of the estimator is sensor dependent. We use median filters of different size to provide a preliminary estimate of the true irradiance. The median filter provides a robust preliminary estimate of the true irradiance by significantly filtering the spatial nonuniformity from the frames corrected for readout nonuniformity and it also reduces the spatial signal resolution. This problem is overcome by the least squares component of the proposed algorithm. The preliminary scene estimate obtained from the frames corrected for readout nonuniformity using median filter is denoted $\bar{x}_k(j)$ for $j = 1, 2, \dots, P$ and $k = 1, 2, \dots, K$. Let us now define a vector which uses the preliminary scene estimates rather than the true irradiance for the recursive least squares curve fitting. This vector is given by [9]

$$\bar{\mathbf{x}}_k(j) = [\bar{x}_k^N(j), \bar{x}_k^{N-1}(j), \dots, \bar{x}_k(j), 1]^T. \quad (3.10)$$

3.2.2 Recursive Least Squares NUC Parameter Estimation

We now seek to find the nonuniformity parameters which minimize the following objective function[9]

$$\epsilon_n(j) = \sum_{i=0}^n \lambda^{n-i} |\hat{y}_i(j) - \mathbf{a}_n(j)^T \bar{\mathbf{x}}_i(j)|^2. \quad (3.11)$$

Table 3.1: Recursive least squares nonuniformity parameter estimation, performed for each pixel j .

1. *Initialization:* Let $\hat{\mathbf{a}}_0(j) = \mathbf{0}$ and $\mathbf{P}_0(j) = \delta^{-1}\mathbf{I}$

2. *Recursive Computations:* For $n = 1, 2, \dots$

$$\mathbf{h}_n(j) = \mathbf{P}_{n-1}(j)\bar{\mathbf{x}}_n(j)$$

$$\mathbf{g}_n(j) = \frac{1}{\lambda + \bar{\mathbf{x}}_n^T(j)\mathbf{h}_n(j)}\mathbf{h}_n(j)$$

$$\mathbf{P}_n(j) = \frac{1}{\lambda}[\mathbf{P}_{n-1}(j) - \mathbf{g}_n(j)\mathbf{h}_n^T(j)]$$

$$\alpha_n(j) = \hat{y}_n(j) - \mathbf{a}_n^T(j)\bar{\mathbf{x}}_n(j)$$

$$\hat{\mathbf{a}}_n(j) = \hat{\mathbf{a}}_{n-1}(j) + \alpha_n(j)\mathbf{g}_n(j)$$

Here λ is a “forgetting” factor allowing the objective function to adapt to drift in the nonuniformity parameters. For $\lambda < 1$, the objective function will be weighted towards the most recent observations. The estimate of the nonuniformity parameters after n frames is then given by

$$\hat{\mathbf{a}}_n(j) = \arg \min_{\mathbf{a}_n(j)} \sum_{i=0}^n \lambda^{n-i} |\hat{y}_i(j) - \mathbf{a}^T \bar{\mathbf{x}}_i(j)|^2, \quad (3.12)$$

where \mathbf{a} is an $N \times 1$ vector and

$$\hat{\mathbf{a}}_n(j) = [\hat{a}_{n,1}(j), \hat{a}_{n,2}(j), \dots, \hat{a}_{n,N+1}(j)]^T \quad (3.13)$$

contains the individual nonuniformity parameter estimates. We minimize the objective function for each frame n using a recursive least squares (RLS) algorithm. Note that we are performing a curve fit between the preliminary scene estimate and the corresponding frame corrected for readout nonuniformity, for each pixel j . The RLS

algorithm allows an estimate to be formed quickly and this estimate is efficiently improved as the incoming frames are acquired. We use a standard RLS algorithm which is summarized in Table 3.1, where $\mathbf{0}$ is an $(N + 1) \times 1$ vector of zeros and \mathbf{I} is an $(N + 1) \times (N + 1)$ identity matrix. In the RLS algorithm, the nonuniformity parameters are initialized to 0. The inverse autocorrelation matrix $\mathbf{P}_0(j)$ is initialized as $\delta^{-1}\mathbf{I}$. Here we use $\delta = .01$. The error between the desired output and the estimated output is formed by applying the previous set of nonuniformity parameters to the observation vector. When the difference is very small the nonuniformity parameters are close to their optimal values and only a small correction is required. However, when the difference is large, the current set of nonuniformity parameters require a large correction.

3.3 Final Nonuniformity Correction

Once the nonuniformity parameters are estimated, the observed frame can be corrected. For $N = 1$, the corrected pixels are given by[9]

$$\hat{x}_n(j) = \frac{\hat{y}_n(j) - \hat{a}_{n,2}(j)}{\hat{a}_{n,1}(j)}. \quad (3.14)$$

For higher order polynomial models, $\hat{x}_n(j)$ is the solution to the polynomial

$$\hat{a}_{n,1}(j)x^N + \hat{a}_{n,2}(j)x^{N-1} + \cdots + \hat{a}_{n,N+1}(j) = \hat{y}_n(j). \quad (3.15)$$

Since multiple roots will result, we select the root closest to the pixel value $\hat{y}_n(j)$.

CHAPTER 4

PERFORMANCE ANALYSIS OF NUC ALGORITHM

In this chapter, we evaluate the performance of the proposed NUC algorithm. The efficacy of the proposed NUC algorithm is demonstrated by applying it to simulated data and infrared imagery. The remainder of this chapter is organized as follows: Section 4.1 presents the results obtained by applying the NUC algorithm to simulated data. In this section quantitative analysis is also performed by computing the mean absolute error and signal to noise ratio. Section 4.2 presents the results obtained using true infrared images. The Matlab source code used to generate the results presented in this chapter is shown in Appendix A.

4.1 Simulated Nonuniformity

The performance of the proposed algorithm is evaluated subjectively and quantitatively by applying it to simulated data. The simulated image sequence is formed from a visible 8-bit image. A 256×256 window is sectioned from the full visible image shown in Fig. 4.1 and the window is translated horizontally and vertically to simulate motion and generate a 100 frame image sequence. The channel level nonuniformity is introduced into each of the four channels configured with the same readout pattern as the Night conqueror infrared camera shown in Fig. 2.1(a). The nonuniformity in each

Table 4.1: Mean and standard deviation (std) for the gain and the bias for each of the four channels to simulate the channel nonuniformity.

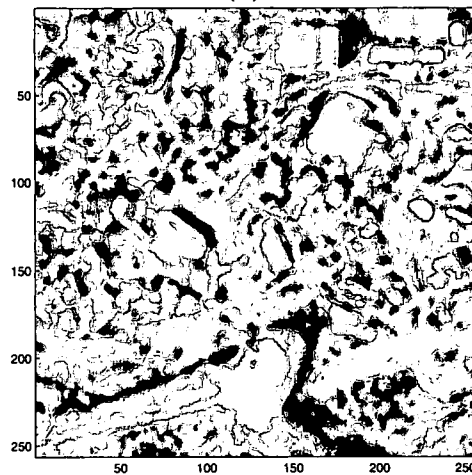
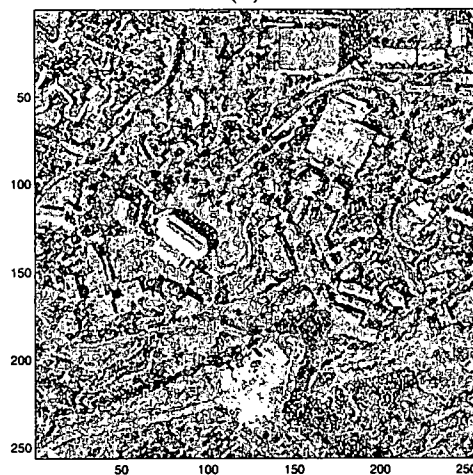
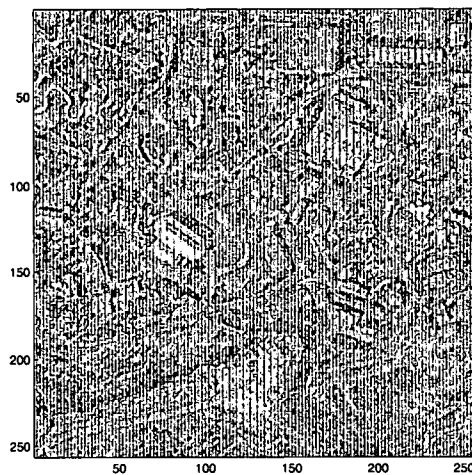
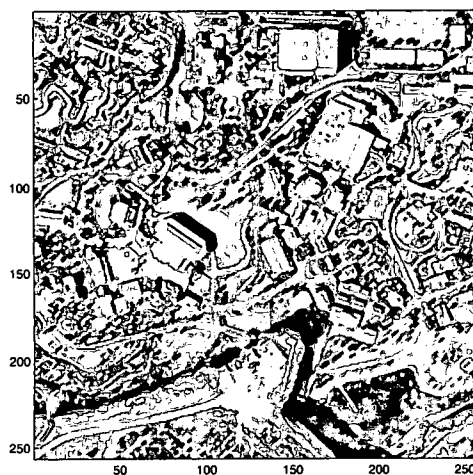
Channel No.	Gain		Bias	
	Mean	Std	Mean	Std
Channel 1	1.25	0.15	0	30
Channel 2	1.5	0.25	0	36
Channel 3	1.75	0.35	0	42
Channel 4	2	0.45	0	48



Figure 4.1: True visible 8-bit image.

of the four channels is simulated using artificial gain and bias modelled as Gaussian random variables. The mean and the standard deviation for the gain and the bias for each of the four channels are chosen such that the level of nonuniformity introduced is of the same order as in real infrared data. The standard deviation of the bias for each channel is proportional to the standard deviation of the raw data. The mean and standard deviation for the gain and the bias for each channel is listed in Table

4.1. The detector level nonuniformity is simulated using an artificial gain modelled as a Gaussian random variable with a mean of one and a standard deviation of 0.02, and an artificial bias modelled as zero mean Gaussian random variable with a standard deviation of 25 (half of the standard deviation of the raw data). The 100'th true



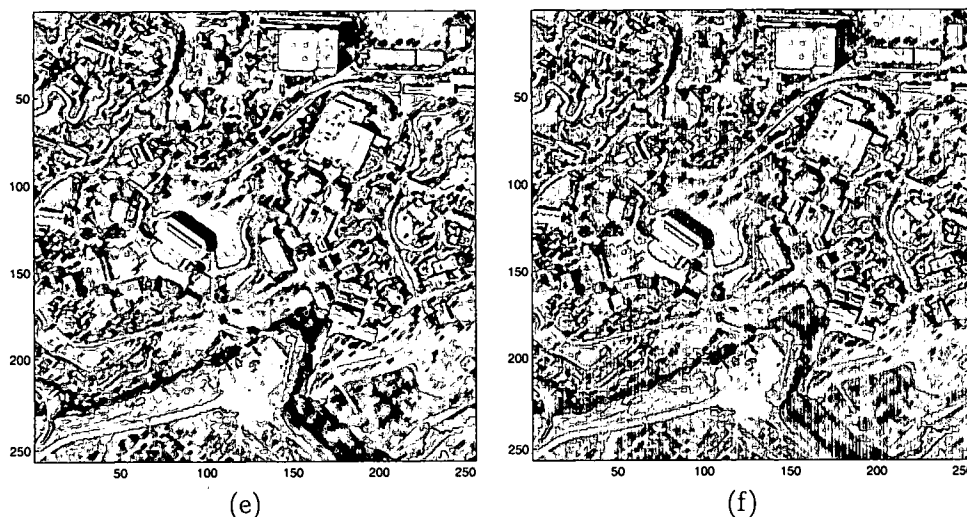
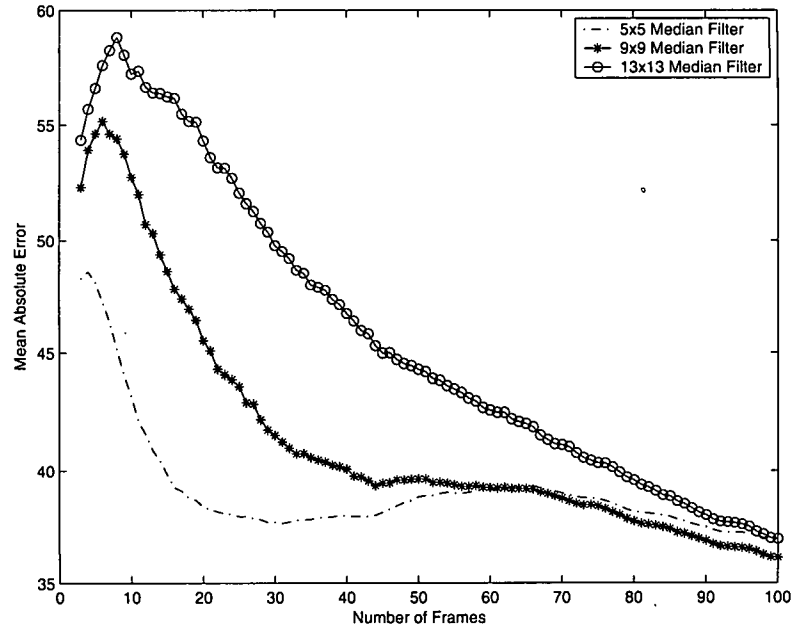


Figure 4.2: Frame 100 in simulated image sequence (a) Uncorrupted true frame, (b) Corrupted with simulated channel and detector nonuniformity, (c) Corrected for readout nonuniformity, (d) Preliminary scene estimate using 5×5 median filter, (e) Corrected using RLS algorithm with 100 frames, (f) Corrected using RLS algorithm without applying readout correction.

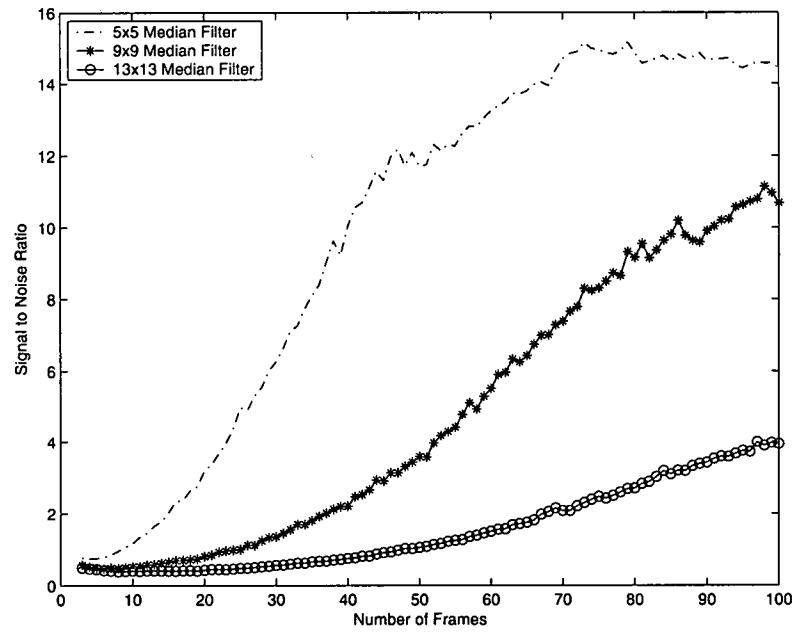
frame is shown in Fig. 4.2(a) and the corresponding corrupted frame from the simulated image sequence is shown Fig. 4.2(b). Note that the difference between the two images is the stripping pattern due to the readout electronics and the spatial noise due to the individual detectors. The image shown in Fig. 4.2(b) was generated by the Matlab script files *addchannoise.m* and *detectornu.m*. The proposed NUC algorithm was applied to the 100 frame image sequence and the channel nonuniformity corrected frame obtained using the readout based NUC is shown in Fig. 4.2(c). This image clearly shows the removal of the stripping patterns caused by the readout electronics and it is corrupted with the spatial nonuniformity. We use 100 frames ($R = 100$) for the readout nonuniformity correction. The preliminary scene estimate obtained by applying a 5×5 median filter to the channel nonuniformity corrected image is shown

in Fig. 4.2(d). The spatial nonuniformity is significantly reduced in this image with a considerable loss in the spatial resolution. A similar estimate is formed for all the preceding frames and these are used in the RLS algorithm to obtain the nonuniformity parameters for each pixel. Here we used a linear model $N = 1$ with $\lambda = 1$. Figure 4.2(e) shows the corrected frame using proposed algorithm. It is observed that the RLS corrected image retains the sharpness of the original and clearly shows reduced nonuniformity. The corrected frame is much improved over the preliminary estimate because it is making use of all 100 frames through the RLS algorithm. Figure 4.2(f) shows the corrected frame obtained by applying the RLS algorithm to the raw frames without the readout based NUC. It is observed that the nonuniformity is not completely removed and a larger size median filter is required to yield a better quality image. The readout based NUC eliminates the use of a larger size median filter and avoids computation complexity. The images shown in Fig. 4.2(c)-(e) were generated by the Matlab script files *framechanncorr.m*, *med2d.m* and *reculrsstda.m* respectively.

The size of median filter is varied and the performance of the proposed algorithm is evaluated by means of two error metrics. The RLS algorithm is applied to the simulated image sequence using a 5×5 , 9×9 and 13×13 median filter and in each case the Mean Absolute Error (MAE) and the Signal to Noise Ratio (SNR) is computed over all the frames in the image sequence. The MAE is defined as the average of the absolute difference between the true frame pixel and the corresponding corrected frame pixel. The SNR is defined as the ratio of the sample variance of the true frame pixel to the sample variance of the error between the true frame pixel and the corresponding corrected frame pixel. Figure 4.3(a) shows the plot of MAE as



(a)



(b)

Figure 4.3: Error metrics to evaluate the performance of the proposed algorithm as a function of median filter size. (a) MAE, (b) SNR.

a function of the number of frames. The MAE is plotted from frame number three, because the RLS algorithm requires a minimum of two frames to provide a meaningful correction. It is observed that the MAE increases for the first few frames and tends to decrease as more incoming frames are acquired. It is observed that the 5×5 median filter has the least error compared to the 9×9 and 13×13 median filter up to frame number 60. The MAE error for the 5×5 median filter decreases rapidly compared to the 9×9 and 13×13 filter. A 5×5 median filter provides a preliminary estimate with less loss in the spatial resolution compared to the 9×9 and 13×13 median filter. Hence, a 5×5 median filter has the potential for faster convergence. These results are generally consistent with the SNR plot shown in Fig. 4.3(b). Larger SNR values indicate better performance. The error plots were generated using the Matlab script file *runsimnuc.m*.

4.2 Real Infrared Data

The proposed algorithm is tested by applying it to two sets of real infrared data.

4.2.1 Amber Infrared Imagery

The first set of real infrared data was acquired using the forward looking infrared FLIR camera that uses a 128×128 Amber AE-4128 infrared FPA with the readout architecture as shown in Fig. 2.1(b). The FPA is composed of Indium-Antimonide (InSb) detectors with a response in the $3\mu\text{m} - 5\mu\text{m}$ wavelength band. This system has square detectors of size $a = b = .040\text{mm}$. The FLIR camera is equipped with a lens aperture of 100mm in diameter and a fnumber of $f/3$. This data was provided by the Sensors Technology Branch at the Air Force Research Laboratories, Wright Patterson Air Force Base. Figures 4.4(a-d) show the results obtained by applying the proposed

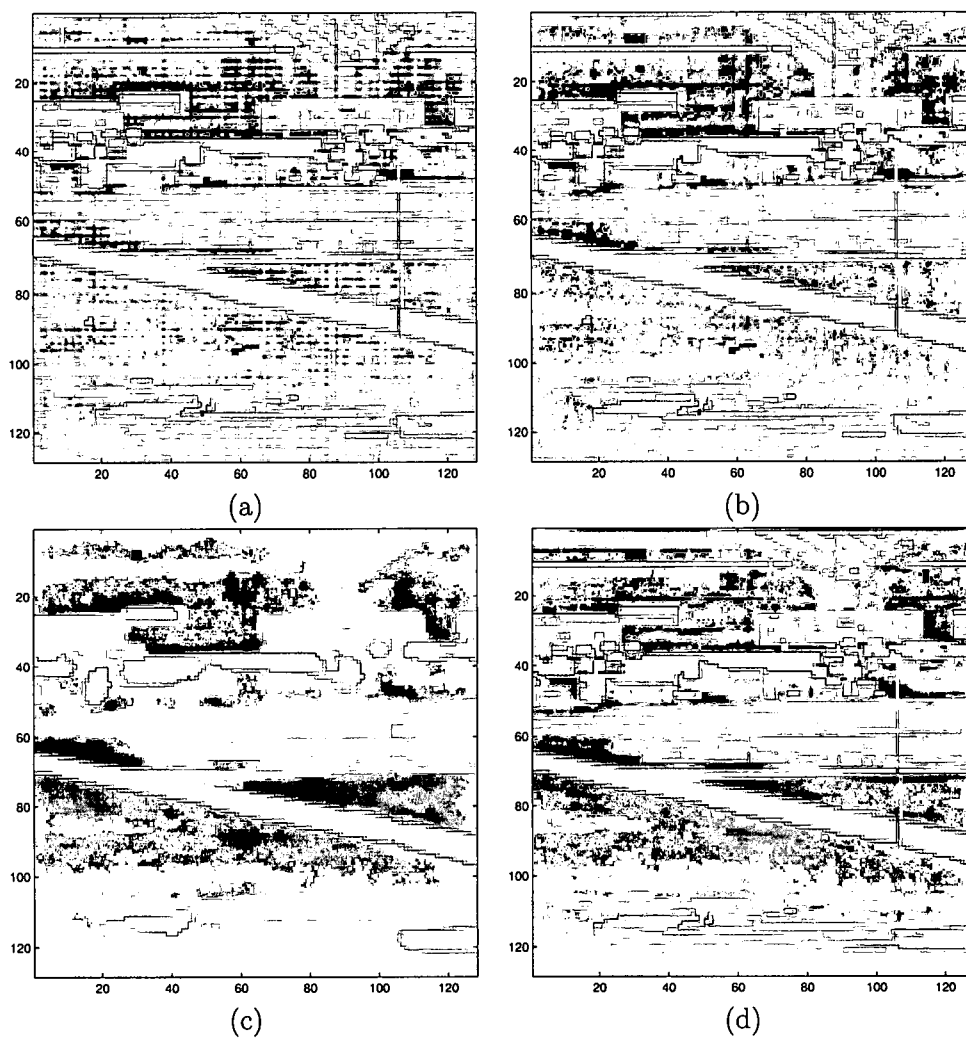


Figure 4.4: Frame 100 in real infrared image sequence acquired using Amber infrared imager. (a) Raw frame, (b) Corrected for readout nonuniformity, (c) Preliminary scene estimate using 3×3 median filter, (d) Corrected using RLS algorithm with 100 frames.

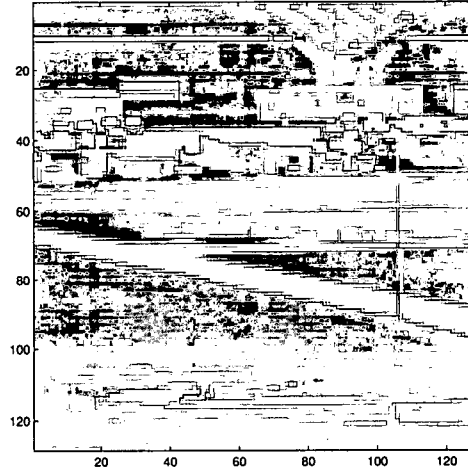


Figure 4.5: Frame 100 of Amber infrared image sequence corrected using RLS algorithm without applying readout correction.

algorithm to infrared image sequence acquired using the Amber infrared FPA. The 100'th raw frame in the Amber infrared image sequence is shown in Fig. 4.4(a). The stripping patterns in the raw frame is similar to the readout architecture of the Amber infrared FPA. The corresponding channel nonuniformity corrected frame is shown in Fig. 4.4(b). The stripping effects are less significant after the channel nonuniformity correction in the frame of Fig. 4.4(b). Note that the channel level correction requires as little as one frame for correction because each group contains sufficient number of pixels that are spatially well distributed. The preliminary scene estimate obtained from the channel nonuniformity corrected frame using a 3×3 median filter is shown in Fig. 4.4(c). Note that the spatial nonuniformity is significantly removed with considerable loss in the spatial resolution in the frame of Fig. 4.4(c). Figure 4.4(d) shows the final nonuniformity corrected frame using the RLS algorithm. It is observed that the spatial nonuniformity has been effectively removed using the proposed technique.

The RLS algorithm was applied to the raw frame without performing the readout based nonuniformity correction. In this case, the preliminary scene estimate was obtained from the raw frame using a 3×3 median filter. The corrected frame using the RLS algorithm without the readout based nonuniformity correction is shown in Fig. 4.5. It is observed from the corrected frame of Fig. 4.5 that the nonuniformity is not completely removed and the stripping patterns are less pronounced. It is possible to remove this nonuniformity if the preliminary scene estimate were to be obtained using a large size median filter. But, it is increasingly burdensome and time consuming to use a larger size median filter. The intermediate stage of readout based nonuniformity correction helps to downsize the size of the median filter and avoid the computation complexity.

4.2.2 Night Conqueror Infrared Imagery

The second data set was acquired using the Night conqueror II FLIR camera with the readout architecture as shown in Fig. 2.1(a). The Night Conqueror II infrared imager uses 256×256 InSb focal plane array with square detectors of size $a = b = .030\text{mm}$. The sensitivity of the detectors is in the $3\mu\text{m} - 5\mu\text{m}$ wavelength band and the imager uses $f/4$ optics. This data set was provided by CMC electronics. Figures 4.6(a-d) show the results obtained by applying the proposed algorithm to infrared image sequence acquired using the Night conqueror infrared camera. The 100'th raw frame in the Night conqueror infrared image sequence is shown in Fig. 4.6(a). The difference in stripping patterns between the raw frame of Fig. 4.6(a) and the raw frame of Fig. 4.4(a) is evident due to the difference in the readout architecture in the infrared cameras. The corresponding channel nonuniformity corrected frame is shown

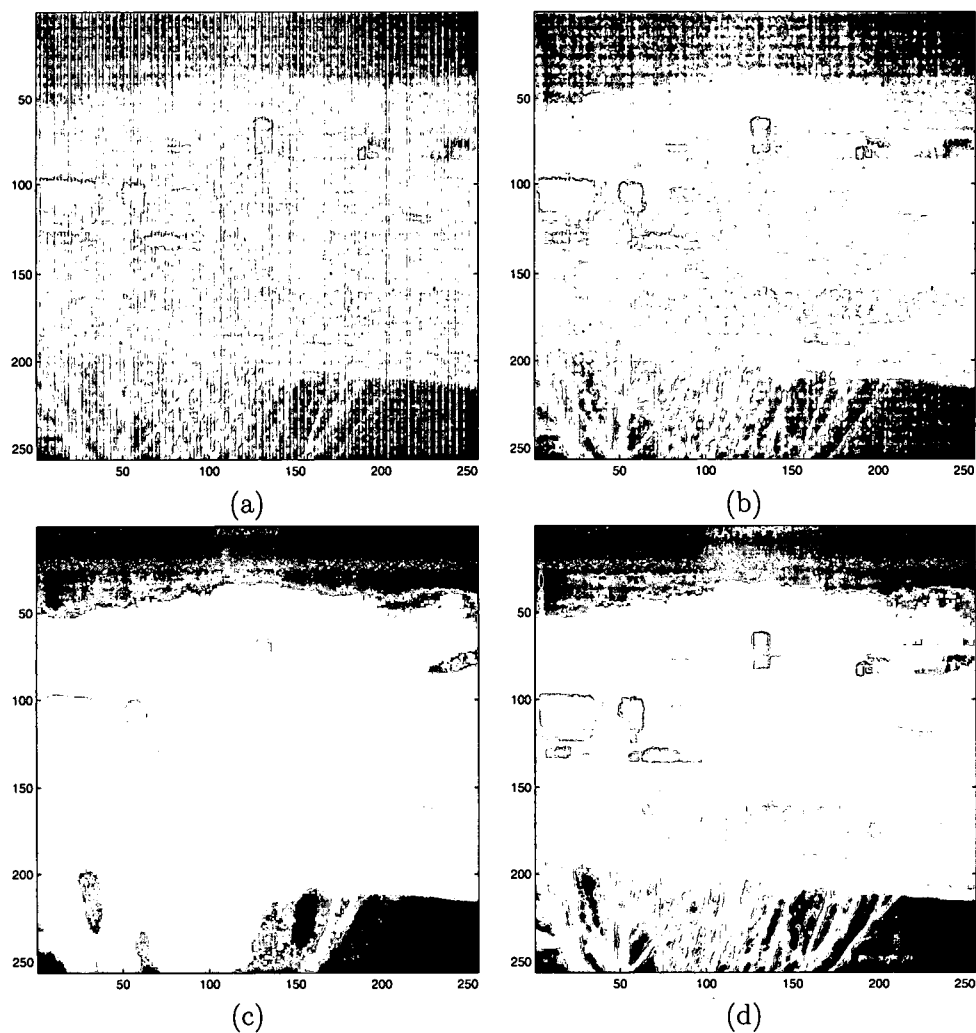


Figure 4.6: Frame 100 in real infrared image sequence acquired using Night conqueror infrared imager. (a) Raw frame, (b) Corrected for readout nonuniformity, (c) Preliminary scene estimate using 13×13 median filter, (d) Corrected using RLS algorithm with 100 frames.

in Fig. 4.6(b). It is also observed that the frame shown in Fig. 4.6(b) has significant level of detector nonuniformity present after channel correction. Hence, a 13×13 median filter was used to provide a preliminary scene estimate. The preliminary scene estimate obtained from the channel nonuniformity corrected frame using a 13×13 median filter is shown in Fig. 4.6(c). Figure 4.6(d) shows the final nonuniformity corrected frame using the RLS algorithm.

CHAPTER 5

REGION-BASED WNN NONUNIFORM INTERPOLATION

We begin this chapter by briefly describing the WNN [40] approach for image superresolution. The WNN SR algorithm [40] is an interpolation-restoration based technique for constructing HR images. The interpolation based SR techniques are simple and computationally less intensive. The observed LR frames are registered with respect to a reference grid. After the registration, the samples from the observed LR frames are inserted into the HR grid. The HR grid contains nonuniformly spaced LR samples and requires a nonuniform interpolation to fully fill the HR grid. The resulting HR image may be subjected to further processing to remove noise and blur. Some of the SR techniques combine the interpolation and restoration in a single step. The quality of the HR image reconstructed using the WNN SR approach is dependent on the distribution of the sub-pixel shifts. The region of HR grid with maximum overlap of LR frames is reconstructed with high quality.

In this chapter, we present a modified version of the WNN SR algorithm to perform HR image reconstruction in different regions of the HR grid based on the overlap of the LR frames. The remainder of this chapter is organized as follows: Section 5.1 presents an overview of the WNN SR algorithm. The modified WNN SR algorithm

is presented in section 5.2. The Matlab source code to generate the figures presented in Section 5.2.1 is shown in Appendix B

5.1 WNN SR Approach

The first step in the WNN SR algorithm [40] is to register the R observed LR frames using a gradient based-registration [22] technique. The motion estimates computed using the registration technique are used to construct the HR image with the weighted nearest neighbor nonuniform interpolation technique. Figure 5.1 shows the

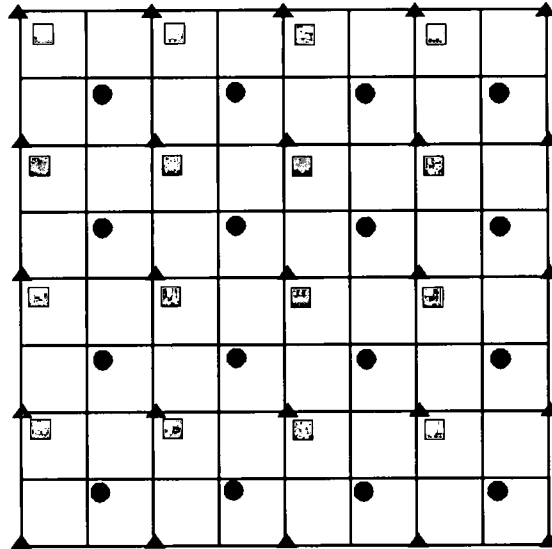


Figure 5.1: HR grid with nonuniformly spaced LR samples.

HR grid with samples from the LR frames. Every pixel on the HR grid is estimated using four nearest frames with weights inversely proportional to the distance between each frame and the pixel. In the HR image reconstruction algorithm, the center of the HR grid is assumed to be located at the position given by average of the shifts.

The reconstructed HR image is processed by a Wiener filter to remove noise and blur incurred during the image acquisition process. With an accurate knowledge of the source of degradation, a Wiener filter can be designed to restore the HR image from blur and noise. An image acquired using an imaging system may be subjected to different forms of degradation. The degradations may be in the form of sensor noise, blur due to camera misfocus, diffraction limited optics, relative object-camera motion, atmospheric turbulence. However, the main source of blurring in detector SR is due to the finite physical dimensions of the detectors on the FPA of the imaging system. The blur resulting from the detectors is caused by the spatial averaging of the light intensity incident on the active region of the detectors. This effect is spatially invariant for a uniform detector array. Thus, the point spread function (PSF) associated with each detector may be expressed as [40]

$$r(x, y) = \frac{1}{ab} \text{rect}\left(\frac{x}{a}, \frac{y}{b}\right), \quad (5.1)$$

where a and b are the dimensions of the individual detectors. The blur caused by the detector charge integration can be modelled as linear convolution of the original continuous image with the PSF given by Equation (5.1).

5.1.1 Wiener Filtering

The reconstructed HR image can be restored from blur and noise using a Wiener filter. A Wiener filter is a linear filter that minimizes the mean squared error (MSE) between the desired image and estimate of the desired image. The continuous frequency response of the Wiener filter is given by [40]

$$H_c^w(u, v) = \frac{H^*(u, v)}{|H(u, v)|^2 + [s_{\eta\eta}(u, v)/s_{ff}(u, v)]}, \quad (5.2)$$

where $*$ denotes the complex conjugate operation, $s_{\eta\eta}(u, v)$ and $s_{ff}(u, v)$ are the power spectral densities of the noise and the original image respectively, and $H(u, v)$ is the detector transfer function or the Fourier transform of the PSF given by 5.1. The blur resulting from the diffraction limited optical system can be incorporated by multiplying the optical transfer function of the optics with the $H(u, v)$. In this application, we neglect the effects of optics. Equation (5.2) is not suitable for practical applications because it is difficult to characterize the noise and moreover the power spectral density of the original image is unknown, since true image is unavailable. Hence, $H_c^w(u, v)$ can be approximated as

$$H_c^w(u, v) = \frac{H^*(u, v)}{|H(u, v)|^2 + nsr}, \quad (5.3)$$

here $nsr = s_{\eta\eta}(u, v)/s_{ff}(u, v)$ is a constant representing the noise-to-signal (NSR) ratio. The HR image reconstruction and the restoration is performed on the sensor's discrete output image. This requires the continuous Wiener filter frequency response given by (5.2) to be mapped to the discrete Wiener filter frequency response in order to perform the restoration with a discrete Wiener filter.

5.2 Modified WNN HR Image Reconstruction

The performance of the WNN SR technique is dependent on the sub-pixel shifts. . This image was generated using Matlab script file *nonuniforminterpolation.m*. When there is large motion between the frames, the overlap of the LR frames on the HR grid is reduced, which gives rise to undesirable border effects. This can be easily verified from the image shown in Fig. 5.2. The borders of the HR image are significantly corrupted because the WNN SR technique assumes that all the LR frames overlap completely with the HR grid. In order to solve this problem, we modify the WNN SR

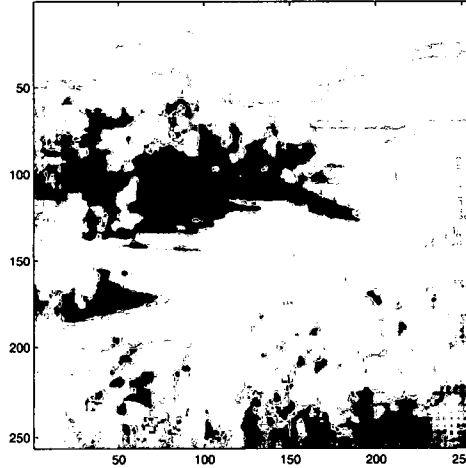


Figure 5.2: HR image reconstructed using WNN SR algorithm

algorithm such that the HR image reconstruction is performed based on the overlap of the LR frames. The flowchart of the modified version of WNN SR algorithm is shown in Fig. 5.3. As in [40] the R LR frames are registered and using the shift estimates the region of overlap of all the R LR frames on the HR grid is identified and reconstructed using WNN SR algorithm. The area of overlap of each LR frame with the HR grid is computed and the frame that constitutes the least area of overlap is identified and eliminated. With the elimination of the LR frame contributing least area of overlap with HR grid, the WNN SR algorithm is used to fill the region, within the overlap of the remaining frames with the HR grid, excluding the previous reconstructed region. Figures 5.4(a) and (b) illustrate the HR image reconstruction process using the modified WNN SR algorithm based on the area of overlap of the LR frames. This process is continued until the number of frames used for reconstruction is less than the number of the neighbors.

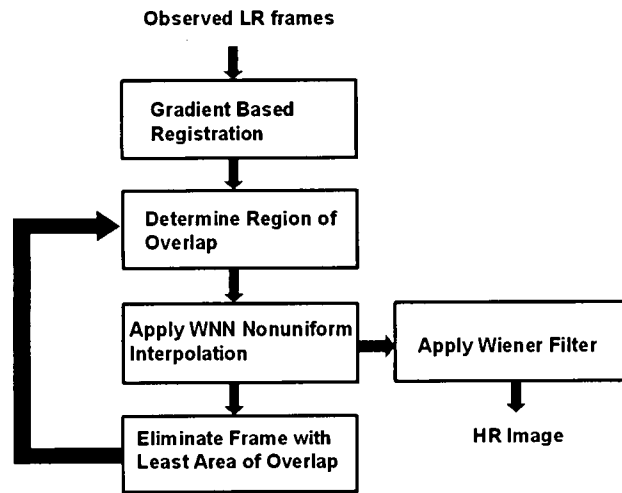


Figure 5.3: Flowchart for modified WNN SR algorithm.

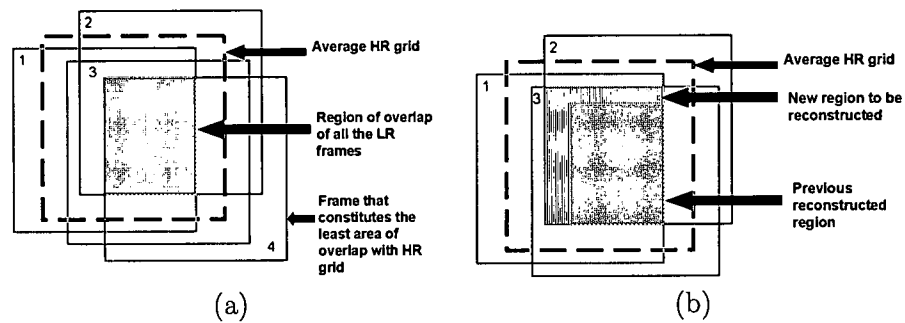


Figure 5.4: (a) HR image reconstructed in the region of overlap on the HR grid (b) new region reconstructed after eliminating frame 4 that contributes the least area of overlap with the HR grid.

5.2.1 Results

The LR frames considered for HR image reconstruction were acquired using an infrared imaging system. A total of 100 LR frames were available in the acquired infrared sequence. Each frame consists of 64×64 pixels. Out of the 100 LR frames, we considered a set of $R = 16$ frames for the reconstruction of one HR image. Frame

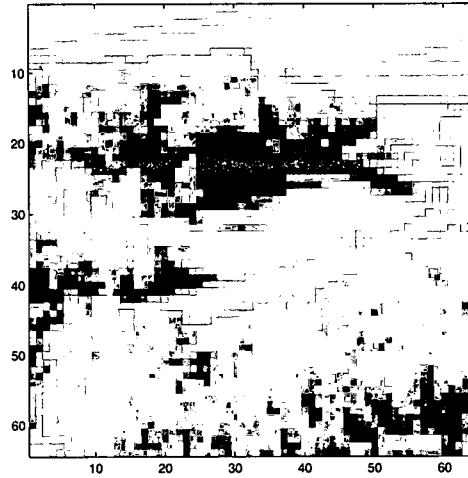


Figure 5.5: Frame 1 from the low resolution infrared sequence.

1 of the LR sequence is shown in Fig. 5.5. The set of LR frames were input to the gradient-based registration algorithm. The computed shift parameters are shown in Fig. 5.6. We considered to increase the resolution by a factor of 4 along the vertical and horizontal direction. The reconstructed HR image using WNN SR algorithm is shown in Fig. 5.2. We used 4 neighbors to fill the each pixel in the HR grid. The quality of the reconstructed image is improved in terms of resolution compared to the LR image shown in Fig. 5.5. It is observed that the reconstructed HR image contains

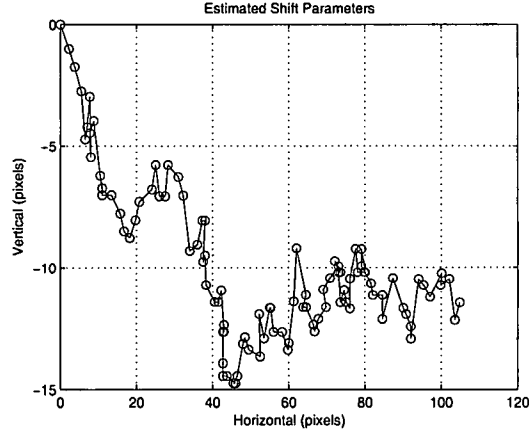


Figure 5.6: Horizontal and vertical shifts in pixels for each low resolution frame.

some noticeable artifacts along the borders. These artifacts are due to the large motion between the LR frames resulting in reduced overlap of the LR frames with the HR grid. The artifacts are more significant along the borders in the horizontal direction compared to the borders along the vertical direction. This is substantiated by the large horizontal shifts as compared to the vertical shift shown in the plot of Fig. 5.6. The WNN algorithm assumes that all the LR frames completely overlap on the HR grid and artificial data is padded to fill up a pixel on the HR grid, where there is no information from those LR frames that are used in the reconstruction process. The HR image reconstructed using the WNN algorithm is processed by a Wiener filter to restore it from blur and noise. We neglect the blur from the optics of the system. The Wiener filter frequency response is obtained from Equation (5.3) with the noise-to-signal ratio set as 0.1. The continuous frequency response is mapped to the discrete domain to perform the filtering operation with a discrete Wiener filter. The HR image estimate using WNN SR algorithm after Wiener filtering is shown in

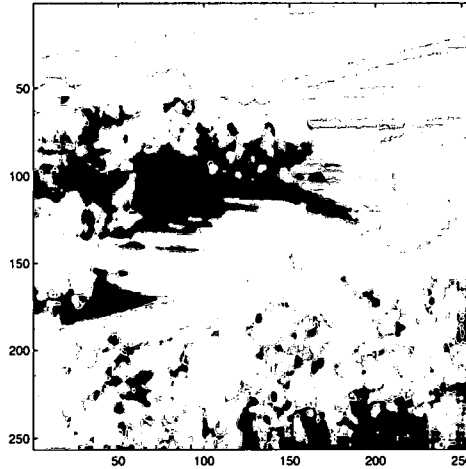


Figure 5.7: HR image estimate using WNN SR algorithm after Wiener filtering.

Fig. 5.7. It is observed that the artifacts are subdued after the filtering process. This image was generated using the Matlab script file *wienerfilter.m*. The HR image using modified WNN SR algorithm before and after Wiener filtering is shown in Fig. 5.8(a) and Fig. 5.8(b) respectively. The maximum and the minimum number of neighbors used to reconstruct the HR image shown is 4. It is inferred that the artifacts are removed in Fig. 5.8(a) compared to the image in Fig. 5.6. But the pixels along the border of the reconstructed HR image shown in Fig. 5.8(a) are not filled. This is because all the 16 frames are not used in the reconstruction process. Figure 5.9(a) shows the image reconstructed by filling only the region on the HR grid where all the 16 LR frames overlap. It can be observed that the region where all the LR frames overlap is reconstructed with high quality but the number of pixels along the border that is not filled is large. However, when the minimum number of neighbors is changed to 1 then the width of the unfilled border is reduced with a compromise

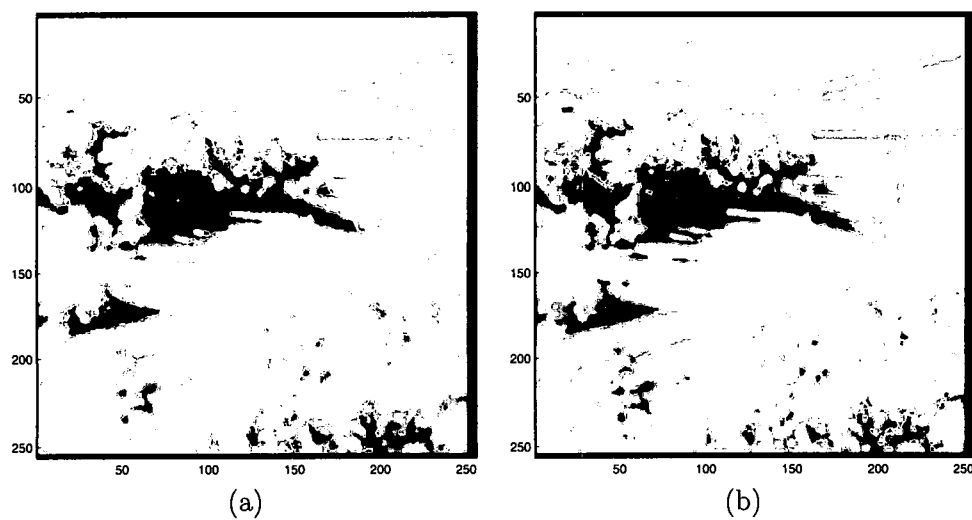


Figure 5.8: HR image estimate using modified WNN SR algorithm with maximum number of neighbor=4 and minimum number of neighbor=4 (a) before Wiener filtering (b) after Wiener filtering.

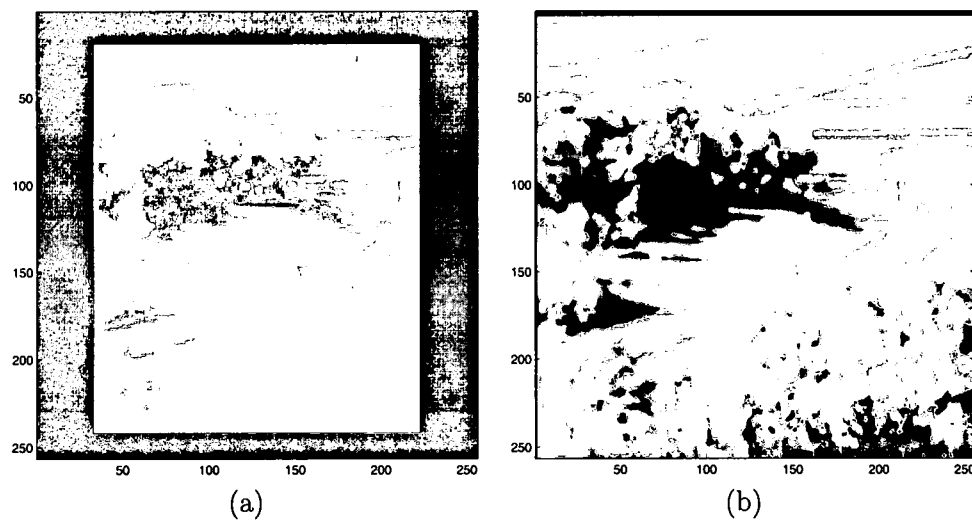


Figure 5.9: (a) Region reconstructed using modified WNN SR algorithm with all the 16 LR frames (b) Region reconstructed using modified WNN SR algorithm with minimum number of neighbor=1.

on the quality. The HR image reconstructed using 1 minimum neighbor is shown in Fig. 5.9(b). The modified WNN SR algorithm is implemented using Matlab script file *fastbordersuperresolution.m*.

CHAPTER 6

PARTITION-BASED SUPERRESOLUTION FOR VIDEO PROCESSING

In chapter 5, we described the Wiener filter used in WNN SR [40] algorithm for removing blur and noise from the HR image. The Wiener filter is designed such that the MSE between the desired image and estimate of the desired image is minimum. This is true when the signal and noise are jointly Gaussian and stationary. However, most natural images are non-stationary. PWS filters have been used for image restoration [4–6] and more recently Shao *et al.* [7] applied PWS filters to the problem of SR enhancement of still images. The optimization of PWS filters has been explored and studied in detail in [44, 45]. The remainder of this chapter is organized as follows. The operation of PWS filters is presented in Section 6.1. The observation model for SR is presented in Section 6.2. Section 6.3 presents the proposed technique to reconstruct HR video sequence from LR video sequence. The notations and definitions introduced in this chapter are meaningful till the end of Chapter 8.

6.1 Partition Weighted Sum Filters

The PWS filters operate with a moving observation window. At each window location, the output is formed using a weighted sum of the pixels within the window.

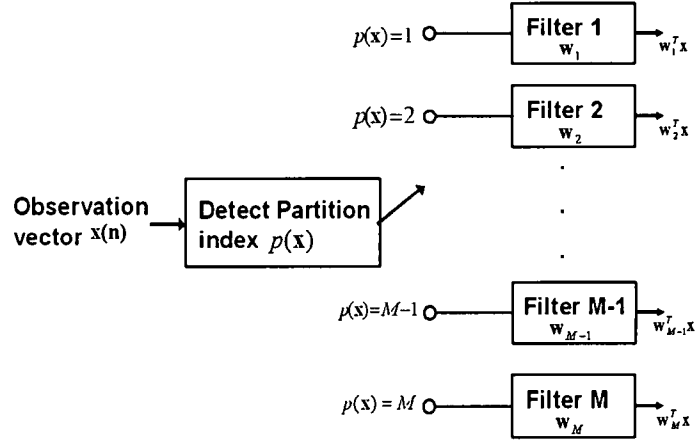


Figure 6.1: Block diagram of PWS filter

The weights are chosen based on the intensity structure of the pixels within the window. The operation of PWS filters is illustrated in Fig. At each window location, the filter determines the underlying structure of the observation vector with the partition index and then applies a weight vector tuned to that structure.

6.2 Super-resolution Observation Model

In this section, we present the general observation model for HR image reconstruction using multiple LR frames. Figure 6.2 illustrates the degradation processes involved during acquisition of the observed LR frames. Each observed LR frame can be regarded as a warped, degraded and subsampled version of the desired HR image. The ideal desired HR image is assumed to be formed by sampling the original continuous scene at or above the Nyquist rate. The desired HR image may be subjected to different kinds of geometrical warping to incorporate global or nonglobal motion in

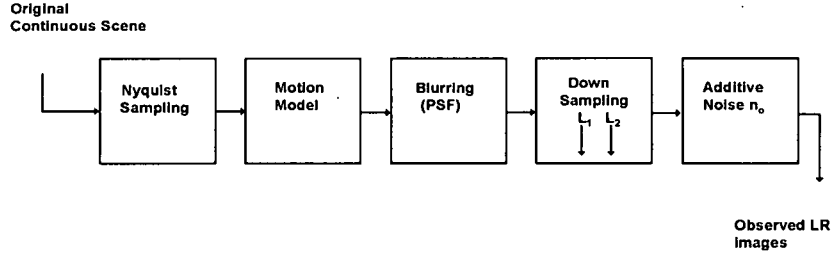


Figure 6.2: Observation model for image superresolution.

the observed LR frames. The common motion models include translation, rotation, affine transformations and general optical flow.

The blurring effects introduced during image acquisition may be caused by a number of phenomena including detector integration, optics, motion, or atmospheric turbulence. In detector SR the dominant source of blur is generally caused by spatial integration of the finite size detectors [1]. The PSF associated with each detector is given by Eq. 5.1. The point spread function can be obtained from experimental measurements or derived based on a model of the optical system [1]. In our observation model, the degraded image is then subsampled to match the effective sampling rate of the detector array. Finally, noise from electronic and photometric sources corrupts the data to produce the modelled observed images. In this paper, we focus on rotational and translational motion, detector blur, and additive Gaussian noise. However, the method presented can be applied with any motion, PSF, and noise model.

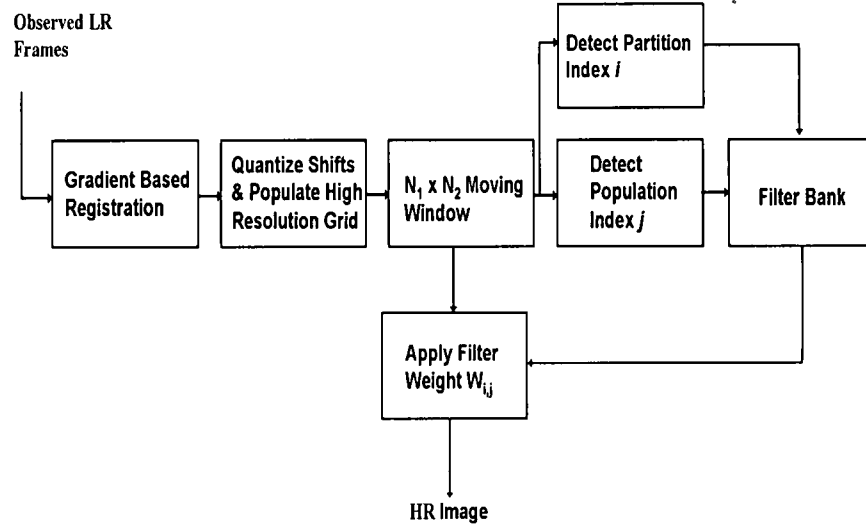


Figure 6.3: Overview of the proposed SR algorithm.

6.3 Proposed Super-resolution Approach

6.3.1 Algorithm Overview

The proposed image superresolution approach is shown in Fig. 6.3. The first step in the HR image reconstruction process is to estimate the motion parameters from the observed LR frames. Different techniques have been used in the past to estimate the motion parameters. We use an iterative gradient-based registration technique [1, 22, 46] to estimate the shifts and rotation from the observed LR frames. The computed shifts are quantized and used to fill the HR grid with the samples from the observed LR frames. The samples from the LR frames are inserted into the nearest pixel position in the HR grid. If pixels from multiple LR frames are placed at the same position in the HR grid, we take an average of these frames. With all the pixels from the LR frames inserted into the HR grid, the resulting HR grid may

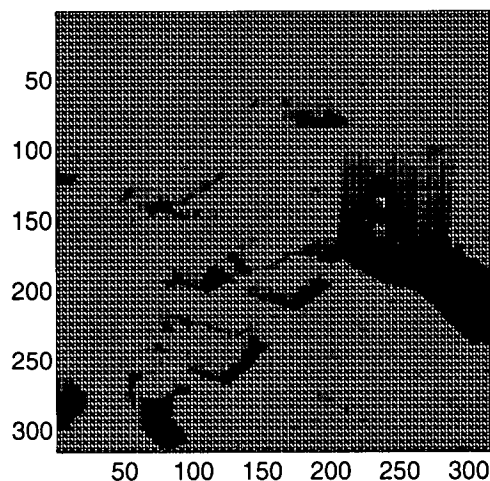


Figure 6.4: Example of a partially populated HR grid. Missing pixels are shown in black.

be populated fully or the grid may be partially populated with some missing pixels. Zeros are inserted into the locations of the missing pixels.

Figure 6.4 shows a partially populated HR grid with non-uniformly spaced pixels obtained from the observed LR frames. As in [7] we use PWS filters to obtain an HR image from this partially populated HR grid. The PWS filters effectively perform nonuniform interpolation and deconvolution simultaneously. The PWS filters use a moving window and form an output at each HR position as a weighted sum of the present pixel values on the HR grid. The weights used depend on the configuration of present pixels in each observation window and the intensity structure of those pixels. We enumerate all possible configurations of present pixels and assign each one a unique population index. The intensity structure in each observation window is classified using VQ and assigned a partition index. Thus, a set of weights is selected

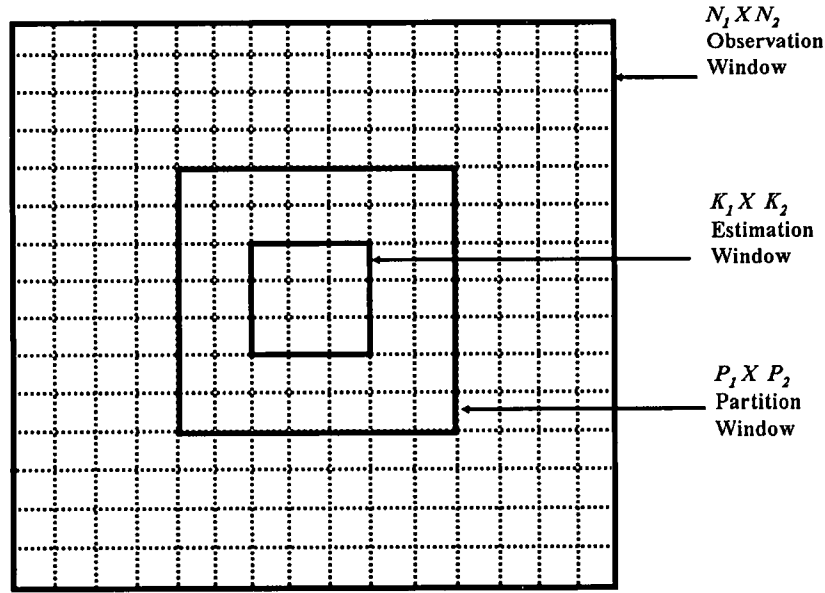


Figure 6.5: $N_1 \times N_2$ local observation window.

to form an estimate of each HR pixel based on the population index and the partition index for the corresponding observation window.

6.3.2 Super-resolution Filters

Consider a sequence of Q LR frames to construct one HR image. Each LR frame consists of $S_1 \times S_2$ pixels. Let the upsampling factors along horizontal and vertical directions be denoted L_1 and L_2 , respectively. The PWS SR filters operate on the partially populated HR grid using an $N = N_1 \times N_2$ moving observation window, as shown in Fig. 6.5. Let the N samples spanned by the observation window be represented by a vector, denoted as

$$\mathbf{x}(\mathbf{n}) = [x_1(\mathbf{n}), x_2(\mathbf{n}), \dots, x_N(\mathbf{n})]^T, \quad (6.1)$$

where $\mathbf{n} = [n_1, n_2]^T$ is the HR image position index corresponding to the center of the moving window. In many cases the observation window will not be fully populated by LR pixels. The location of the observed LR samples and the missing pixels within the window depends on the motion present in the input LR sequence. For video processing, this changes with each new set of input frames. Let us define the set of pixel position indices in the observation window as $\psi = \{1, 2, \dots, N\}$. The set of all subsets, or power set, of ψ can be written as $\phi = 2^\psi$. Note that the power set contains all possible observed configurations of present and missing LR pixels in a observation window. The total number of subsets is given by $|\phi| = 2^N$. Let each of these subsets be enumerated as ϕ_j , for $j = 1, 2, \dots, 2^N$. With this defined, let $\tilde{\mathbf{x}}(\mathbf{n})$ represent the observation vector that contains only the samples in the observation window provided by the LR frames (present samples only). In particular, $\tilde{\mathbf{x}}(\mathbf{n})$ can be written as

$$\tilde{\mathbf{x}}(\mathbf{n}) = [x_i(\mathbf{n}) : i \in \phi_{j(\mathbf{n})}], \quad (6.2)$$

where $j(\mathbf{n})$ is the index corresponding to the observed subset of LR pixels present in the observation window at position \mathbf{n} . Consider a $P = P_1 \times P_2$ partition window centered within the $N_1 \times N_2$ moving observation window as shown in Fig. 6.5. The mean-subtracted samples contained within the partitioning window are written as a vector,

$$\bar{\mathbf{x}}(\mathbf{n}) = [\bar{x}_1(\mathbf{n}), \bar{x}_2(\mathbf{n}), \dots, \bar{x}_P(\mathbf{n})]^T. \quad (6.3)$$

The vector $\bar{\mathbf{x}}(\mathbf{n})$ contains samples from the LR frames and zeros for missing pixels. The samples in this partition window are used to partition the observation space using VQ. A VQ codebook is generated from a suitable training image using the Linde-Buzo Grey (LBG) algorithm [47]. Let this codebook be denoted $C = \{\mathbf{z}_i, i = 1, \dots, M\}$,

where $\mathbf{z}_i = [z_{i,1}, z_{i,2}, \dots, z_{i,P}]^T$ is the i th mean-subtracted codeword. The partition index, assigned to an observation vector, is determined by the partitioning function

$$p(\bar{\mathbf{x}}(\mathbf{n})) = \arg \min_i \|\bar{\mathbf{x}}(\mathbf{n}) - \bar{\mathbf{z}}_i\|^2, \quad (6.4)$$

where $\bar{\mathbf{z}}_i$ is the modified version of the codeword \mathbf{z}_i such that it contains zeros at the positions corresponding to the missing pixels in the vector $\bar{\mathbf{x}}(\mathbf{n})$.

Based on the population index and the partition index, we select a set of weights to form output estimates for the current observation window. Specifically, the output of the PWS SR filter is given by

$$\mathbf{y}(\mathbf{n}) = \mathbf{W}_{p(\bar{\mathbf{x}}(\mathbf{n})), j(\mathbf{n})} \bar{\mathbf{x}}(\mathbf{n}), \quad (6.5)$$

where $\mathbf{y}(\mathbf{n}) = [y_1(\mathbf{n}), y_2(\mathbf{n}), \dots, y_K(\mathbf{n})]^T$ is an output vector corresponding to the $K = K_1 \times K_2$ estimation window shown in Fig. 6.5. Note that instead of forming a single pixel estimate for the observation window, it is computationally advantageous to estimate several output pixel values from each observation window. In this way, the observation window can raster-scan across the HR grid in increments of K_1 pixels vertically and K_2 horizontally (rather than one pixel at a time). The filter weights are contained in the matrix $\mathbf{W}_{i,j}$, where $i = 1, 2, \dots, M$ is the partition index and $j = 1, 2, \dots, 2^N$ is the population index. The weight matrix consists of K rows and q columns, where q is the length of the vector $\bar{\mathbf{x}}(\mathbf{n})$. Each row contains the weights to apply to the present samples in the window to form one output estimate. The optimization of the weights is addressed in the next section. Note that the total number of possible weight matrices required for M partitions is given as $M(2^N - 1)$. Figure 6.6 shows the number of matrices required with $M = 10$ partitions as a function of observation window size, N . For small values of M and N , it is possible

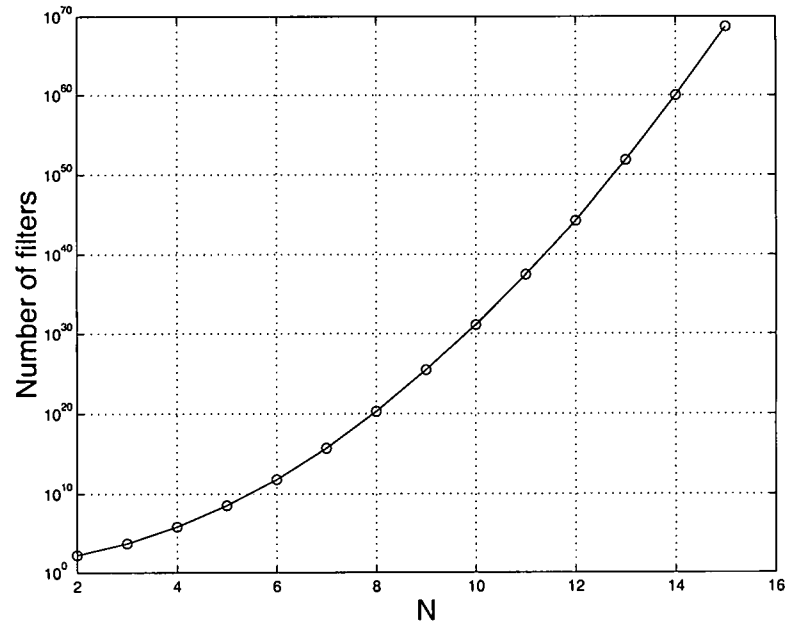


Figure 6.6: Number of PWS SR filters required for $M = 10$ partitions.

to precompute and store the weights. However as N increases, this quickly becomes impractical due to memory limitations and a new methodology is required. This plot was generated using the Matlab script file *numfilters.m*.

CHAPTER 7

OPTIMIZATION AND COMPUTATIONAL COMPLEXITY OF PWS SR FILTERS

In chapter, 6 the PWS SR filters have been defined. In this chapter, we develop a new training procedure that facilitates the application of PWS SR filters to superresolution of video. The computational efficiency of PWS SR filters is analyzed and studied for translational and rotational motion models. The remainder of this chapter is organized as follows. Section 7.1 presents the optimization of the PWS SR filters. A detailed Computational complexity analysis of PWS SR filters is presented in Section 7.2.

7.1 Filter Optimization

The filter weights used in HR image reconstruction are generated similar to that of a FIR Wiener filter. Except here, the weights are generated for every partition, population configuration, and output position. The weights contained in the weight matrix $\mathbf{W}_{i,j}$ are computed using the Wiener-Hopf equations to minimize the mean squared error. This requires the estimation of the autocorrelation and crosscorrelation matrices. Figure 7.1 summarizes the training procedure. A statistically representative desired image is required. The VQ codebook is generated from this image using

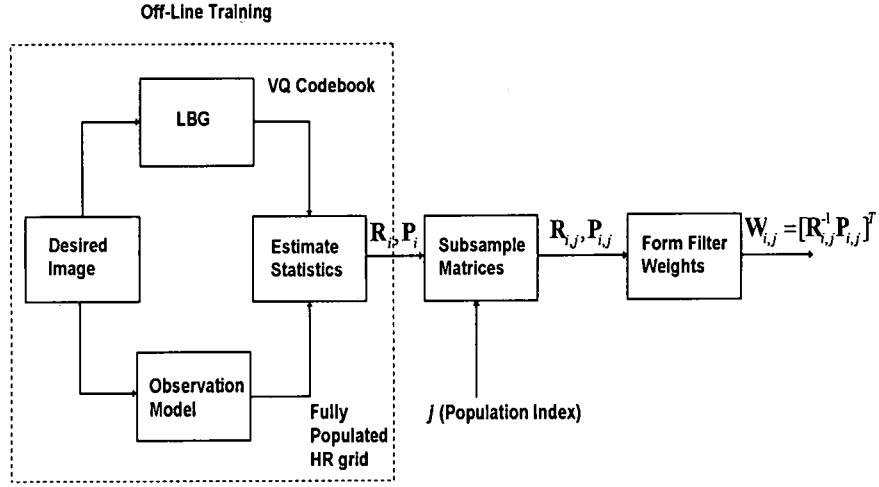


Figure 7.1: Block diagram illustrating the filter optimization procedure.

the LBG algorithm [47]. The knowledge of the autocorrelation and cross correlation matrices is required for computing the filter weights. The statistics can be evaluated using the probability density function of the data. Since the density function is unknown, the statistics is estimated with the ensemble averages of the samples obtained from the training data. The desired image is degraded according to the observation model to generate a fully populated HR grid filled with the simulated LR frames. From this image we can form sample estimates of the autocorrelation matrix $\mathbf{R}_i = E[\mathbf{x}(\mathbf{n})\mathbf{x}(\mathbf{n})^T \mid \bar{\mathbf{x}}(\mathbf{n}) \in \Omega_i]$ for $i = 1, 2, \dots, M$, where Ω_i is the i th partition. Using the degraded and the desired images, we can estimate the crosscorrelation matrix

$$\mathbf{P}_i = \begin{bmatrix} \mathbf{p}_{i,1} & \mathbf{p}_{i,2} & \dots & \mathbf{p}_{i,K} \end{bmatrix}, \quad (7.1)$$

where $\mathbf{p}_{i,k} = E[d_k(\mathbf{n})\mathbf{x}(\mathbf{n})^T \mid \bar{\mathbf{x}}(\mathbf{n}) \in \Omega_i]$ and $\mathbf{d}(\mathbf{n}) = [d_1(\mathbf{n}), d_2(\mathbf{n}), \dots, d_K(\mathbf{n})]^T$ are the desired pixel values corresponding to the pixels in vector \mathbf{y} . The autocorrelation matrix \mathbf{R}_i and the crosscorrelation matrix \mathbf{P}_i are pre-computed off-line for $i = 1, 2, \dots, M$ using the training data.

One of the innovative aspects of this work is that we can now readily determine the weights for any population index as shown in Fig. 7.1. To do so we must form $\mathbf{R}_{i,j}$ and $\mathbf{P}_{i,j}$, which are the subsampled autocorrelation and crosscorrelation matrices for the i th partition and j th configuration, respectively. In particular, the matrix $\mathbf{R}_{i,j}$ consists of elements in \mathbf{R}_i , obtained from the intersection of the rows and columns determined by the indices in the set ϕ_j . The matrix $\mathbf{P}_{i,j}$ is formed from \mathbf{P}_i , by removing the rows determined by the indices in the set ϕ_j .

Now, as with the PWS filter [4], the coefficients in the weight matrix are given by the solution to the Wiener-Hopf equations for the particular partition and configuration

$$\mathbf{W}_{i,j} = [\mathbf{R}_{i,j}^{-1} \mathbf{P}_{i,j}]^T. \quad (7.2)$$

The weight coefficients are optimum in an MSE sense.

7.2 Computational Complexity

In this section, we perform computational complexity analysis of the PWS SR filters. We perform the analysis by calculating the number floating point operations (flops) required for generating one output HR pixel. The important step in the proposed algorithm involves the computation of filter weights given by Eq. (7.2). The weights are computed efficiently using Cholesky factorization. The Cholesky factorization requires $\frac{n^3}{3}$ flops to perform LU decomposition of an $n \times n$ autocorrelation

matrix and $2n^2$ flops to solve the weights using forward and backward substitution. The number of times we need to compute these weights per output frame is a critical factor.

7.2.1 Translational Motion

With only translation motion between the LR frames, the HR grid exhibits a repetitive structure in the region where the LR frames overlap. This means that only a relatively small number of population configurations are observed across the HR entire image. Consequently, only a small number of filter weight matrices need be computed using the method shown in Fig. 7.1. We take advantage of this regularity to estimate the HR image with fewer computations. First, consider the case where the estimation window size is different from the upsampling factor (i.e., $K_1 \neq L_1$ and $K_2 \neq L_2$) as shown in Fig. 7.2(a). Under this condition, there exists $L = L_1L_2$ unique estimation-window population configurations across the HR grid. It should be noted that each of the L configurations repeat every L_1K_1 pixels along the vertical direction and L_2K_2 pixels along the horizontal direction. In this case, the weights are easily pre-computed for the L patterns for each of the M partitions prior to processing the HR grid. The number of flops per output pixel required to implement the PWS SR filters with $M > 1$, $K_1 \neq L_1$ and $K_2 \neq L_2$ is approximately given by

$$flops \cong \frac{(fP)^2 + 2fPM}{K} + \frac{\frac{ML(fN)^3}{3} + 2MLK(fN)^2}{(S_1L_1 - N_1 + 1)(S_2L_2 - N_2 + 1)} + fN, \quad (7.3)$$

where f is the fraction of pixels on the HR grid that are populated. For $f = 1$ the HR grid is completely populated with the LR samples and requires only restoration from blur and noise. The first term in Eq. (7.3) represent the flops required to compute the Euclidian distances for the VQ partitioning. The second term represents the

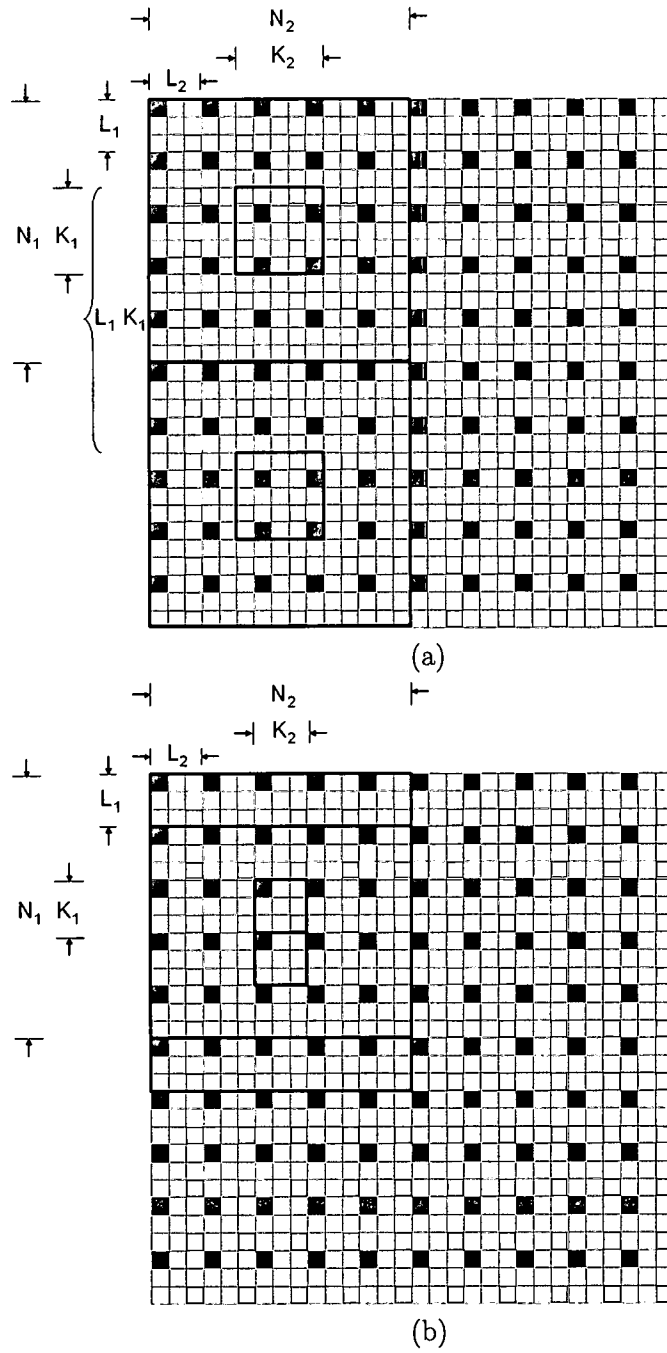


Figure 7.2: $N_1 \times N_2$ moving observation window on HR grid for (a) $L_1 = L_2 = 3$ and $K_1 = K_2 = 5$ (b) $K_1 = L_1 = K_2 = L_2 = 3$.

flops required to solve for the weights for the L patterns. The LU factorization of the autocorrelation matrix is performed only once for each of the L patterns, whereas the forward and backward substitution is carried out for every pixel. The last term represents the flops required to perform the weighted sum to obtain the output.

Next, consider the case where $K_1 = L_1$ and $K_2 = L_2$, as shown in Fig. 7.2(b). In this scenario, there is only one population configuration for all observation windows. The periodicity of population configuration is K_1 and K_2 along the vertical and horizontal directions, respectively. This periodicity corresponds exactly to the observation window step size, providing a nice computational savings. This savings results from only needing to pre-compute one weight matrix for each of the M partitions to processing the HR grid. The flops required to implement the PWS SR filters with $K_1 = L_1$, $K_2 = L_2$, and $M > 1$ is approximately given by

$$flops \approx \frac{(fP)^2 + 2fPM}{K} + \frac{\frac{M(fN)^3}{3} + 2MK(fN)^2}{(S_1L_1 - N_1 + 1)(S_2L_2 - N_2 + 1)} + fN. \quad (7.4)$$

When $M = 1$, the VQ partitioning is not explicitly carried out and the number of flops required is obtained by omitting the first term in Eq. (7.3) and (7.4). Figure 7.3 shows the flop count plotted as a function of f for and $N = 15 \times 15$ observation window, $S_1 = S_2 = 64$ and $M = 1$. This plot was generated using the script file *flopcount.m*. The flop count is plotted for two different cases: $K_1 = L_1 = K_2 = L_2 = 5$ and $K_1 = K_2 = 5$ with $L_1 = L_2 = 7$. It requires fewer flops to estimate the HR image when the HR grid is sparsely populated. The computational complexity increases as more samples are observed within the local window. Furthermore, it is observed from Fig. 7.3 that choosing an estimation window size different from the upsampling factor increases the computational burden. For comparison, the complexity of the weighted nearest neighbor (WNN) approach in [36, 40] is also shown in Fig. 7.3. With WNN,

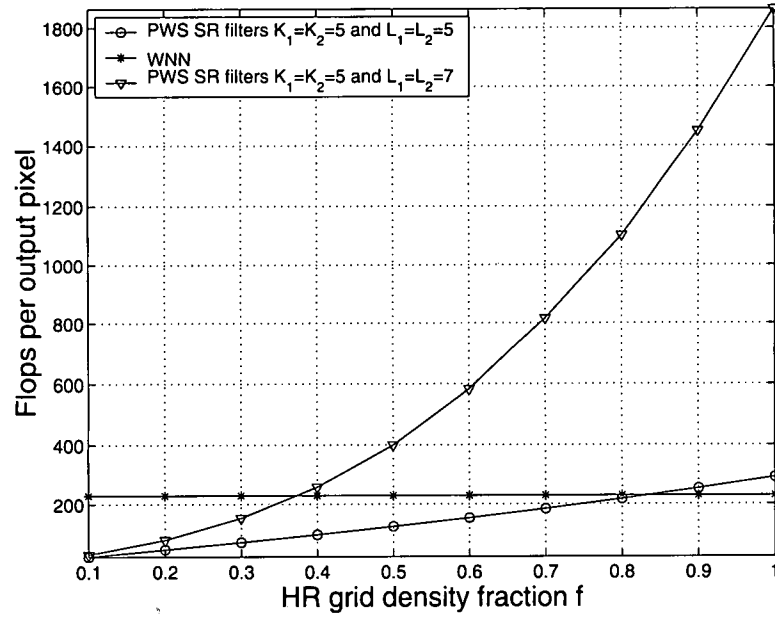


Figure 7.3: Flops per output HR pixel as a function of the HR grid density fraction f for translational motion with $N = 15 \times 15$, $S_1 = S_2 = 64$, and $M = 1$.

every pixel on the HR grid is estimated using four nearest LR pixels with weights inversely proportional to distance from LR pixel to the HR pixel being estimated. A Wiener filter is implemented after filling the HR grid. Note that for all but the highest HR grid densities, the proposed method has lower computational complexity than the WNN. It can be easily verified that PWS SR filters with $M > 1$ involve more computations than $M = 1$ due to the additional burden involved in comparing the local structure to the different partitions and in computing the weight matrix for the different partitions.

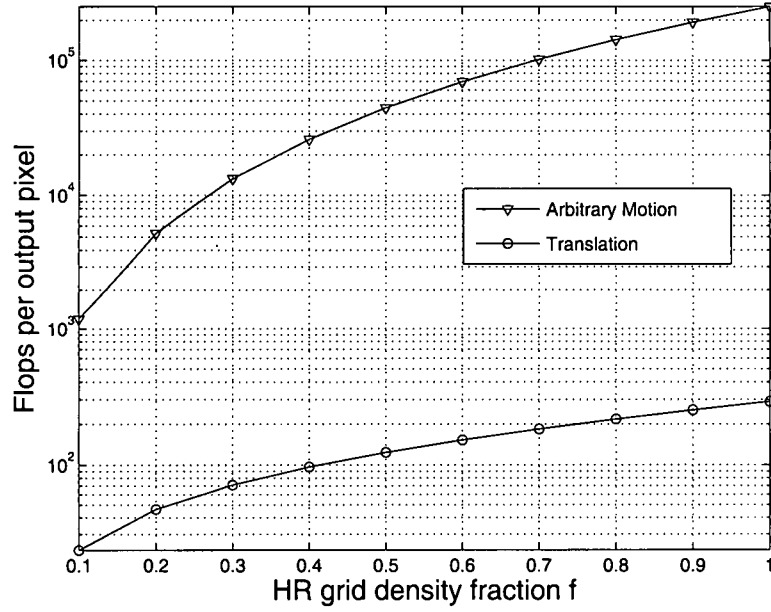


Figure 7.4: Flops per output HR pixel as a function of the HR grid density fraction f for arbitrary motion compared with the computational complexity of translational motion with $N = 15 \times 15$, $S_1 = S_2 = 64$, $K_1 = K_2 = 5$, $L_1 = L_2 = 5$ and $M = 1$.

7.2.2 Arbitrary Motion

When the motion between the LR frames is other than global translational motion, the regularity in the HR grid pattern may not be preserved. In the worst case scenario, a weight matrix needs to be computed for every estimation window in the HR grid. In particular, the LU factorization of the autocorrelation matrix is performed for every estimation window in the HR grid and the forward and backward substitution is carried out for every HR pixel. In this case, the total number of flops per output pixel required is approximately given by

$$flops \cong \frac{(fP)^2 + 2fPM}{K} + \frac{(fN)^3}{3K} + 2(fN)^2 + fN. \quad (7.5)$$

The flop count per output HR pixel is plotted as function of f for arbitrary motion with $N = 15 \times 15$, $K_1 = L_1 = K_2 = L_2 = 5$ and $M = 1$. This plot was generated using the script file *flopcount.m*. It can be inferred from the plot of Fig. 7.4 that going from translational motion to rotation or other type of motion model significantly increases the computational cost.

CHAPTER 8

HR VIDEO RECONSTRUCTION AND RESTORATION USING PWS SR FILTERS

In this chapter, we present the results obtained by applying the PWS SR filters to simulated data and real infrared imagery. The performance of the PWS SR filters is compared to that of: the WNN SR approach [36, 40]; Delaunay triangulation followed by wiener filtering; and the regularized least squares (RLS) image reconstruction algorithm [1]. The Delaunay triangulation technique is implemented using the built-in MATLAB function GRIDDATA. The remainder of this chapter is organized as follows. The results obtained using simulated data is presented in Section 8.1. Quantitative analysis of the PWS SR filters is also presented in this section. The application of PWS SR filters to true infrared video sequence is described in Section 8.2. The Matlab source code used to generate the HR images presented in this chapter is shown in Appendix C

8.1 Simulated Data

The effectiveness of the proposed algorithm is evaluated quantitatively by applying it to simulated low-resolution frames. A visible 8-bit grayscale image of size 320×320 is blurred and downsampled by factor of $L_1 = L_2 = 5$ to yield 25 low resolution

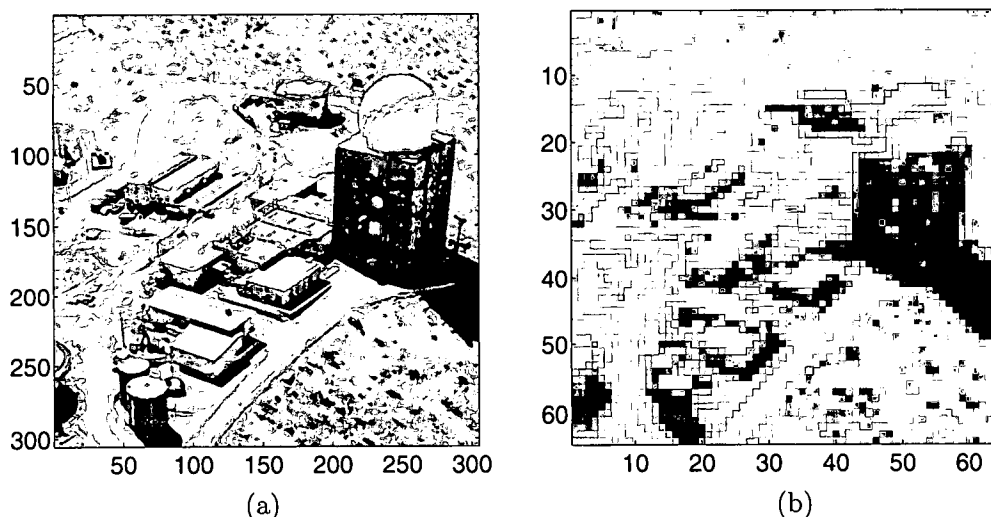


Figure 8.1: Simulation images. (a) Original 8-bit image, (b) simulated noisy low resolution frame.

frames. A 5×5 moving average filter is used to simulate the blur caused by the detector integration. Finally, noise is modelled as a white Gaussian random process with zero mean and variance 10, which is additively introduced in each low resolution frame. The original image and one simulated LR frame are shown in Fig. 8.1.

In order to evaluate the performance of the algorithms with different numbers of frames, we generate several LR sequences. In these sequences, the LR frames are taken from the fully populated HR grid and numbered as shown in Fig. 8.2(a). The different combinations of simulated LR frames used to construct the LR input sequences are shown in Fig. 8.2(b). The exact motion parameters are used by all SR methods so that we may compare image reconstruction performance apart from image registration. We consider SR with $L_1 = L_2 = 5$ and different numbers of LR frames, from $Q = 1$ up to $Q = 25$. The VQ codebook, autocorrelation matrices, and

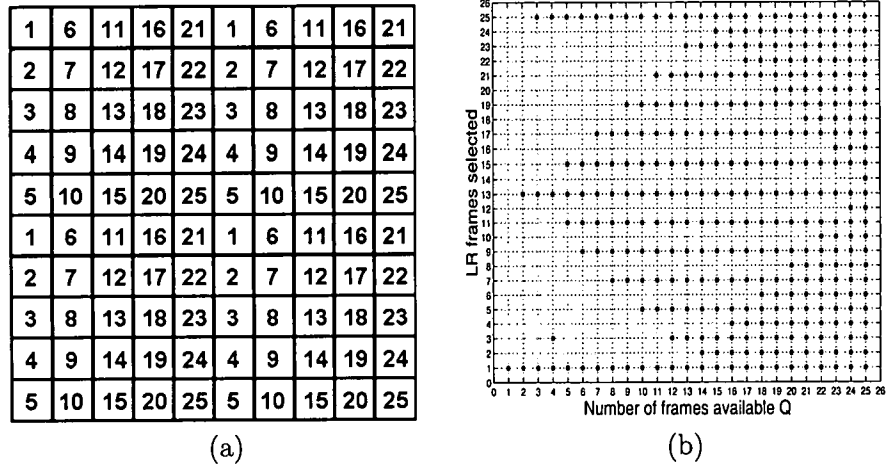


Figure 8.2: Simulation parameters. (a) Position of the simulated LR frames relative to the original HR image, (b) combination of simulated LR frames used to generate the different length input sequences.

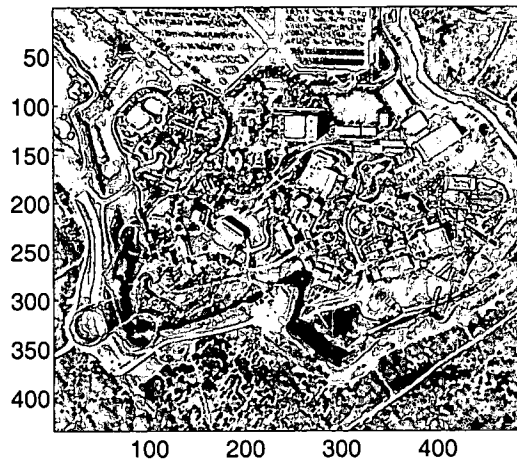


Figure 8.3: 8-bit training image used to obtain VQ codebook, and the PWS autocorrelation and crosscorrelation matrices for the simulation results and for the infrared image results.

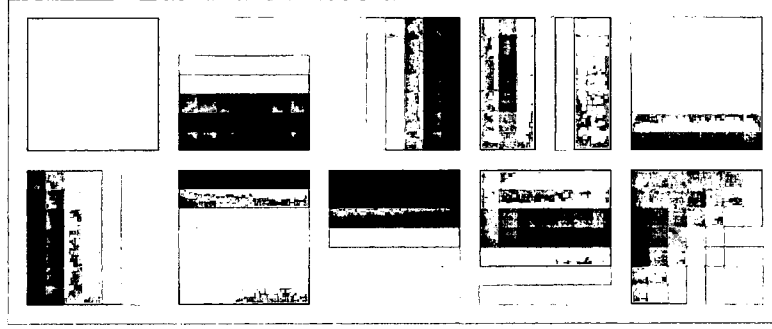


Figure 8.4: VQ codebook for $M = 10$ partitions.

crosscorrelation matrices are obtained from the training image shown in Fig. 8.3. These parameters are used for all of the PWS SR experimental results, including the infrared image results. The VQ codebook obtained using LBG algorithm for $M = 10$ partitions is shown in Fig. 8.4. The autocorrelation matrices for $M = 10$ partitions is shown in Figs. 8.5(a)-(j)

Figure 8.6 shows the plot of MAE as a function of the number of partitions. It is observed that the MAE is minimum for PWS SR filters with $M = 10$. There is no significant difference in the MAE for different partitions. However, as the number of partition increases, the performance of PWS SR filters decrease due to the insufficient training data. Figure 8.7 shows the MAE for several SR methods as a function of the number of observed LR frames, Q . For the PWS SR filters, we employ an observation window size of $N_1 = N_2 = 15$, a partition sub-window of size $P_1 = P_2 = 7$, and an estimation window of size $K_1 = K_2 = 5$. We compare the PWS SR filters with $M = 1$ and $M = 10$. For the WNN and Deluanay triangulation algorithms, we implement the Wiener filter as described in [40] and select the noise-to-signal (NSR) ratio parameter

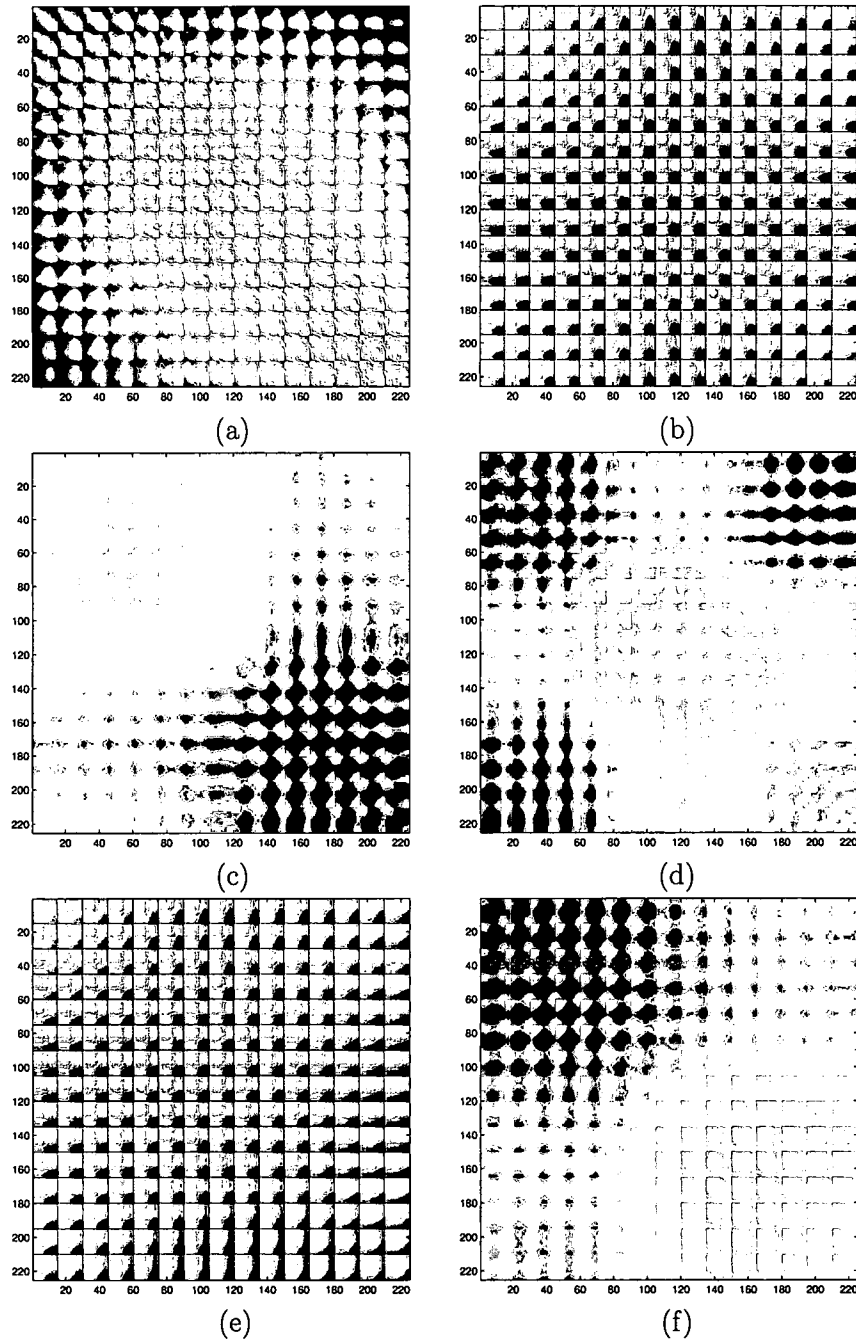


Figure 8.5: Autocorrelation matrices for the different partitions. (a) $M = 1$, (b) $M = 2$, (c) $M = 3$, (d) $M = 4$, (e) $M = 5$, (f) $M = 6$, (g) $M = 7$, (h) $M = 8$, (i) $M = 9$, (j) $M = 10$.

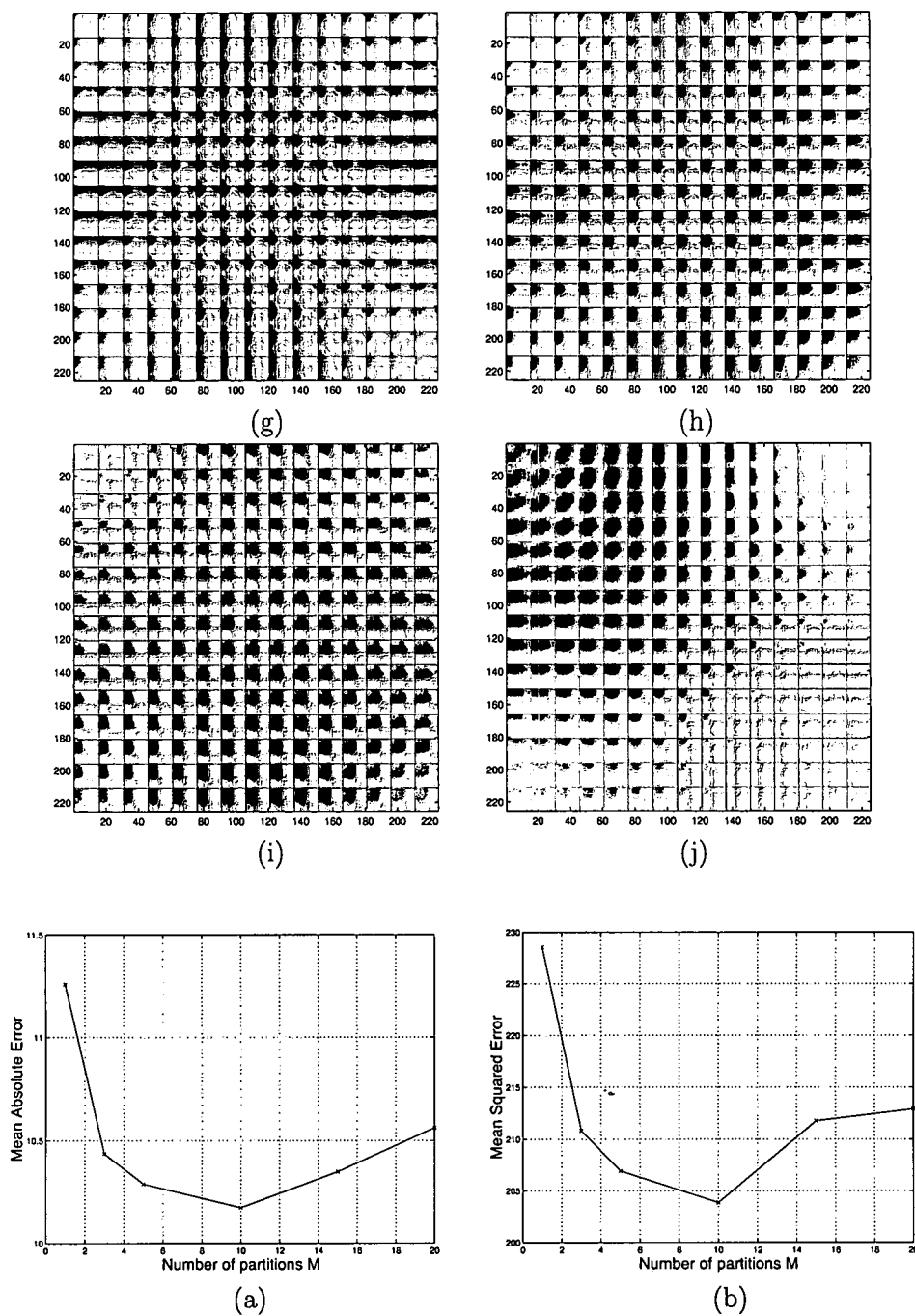


Figure 8.6: Error for PWS SR filter as function of number of partitions (a) MAE (b) MSE.

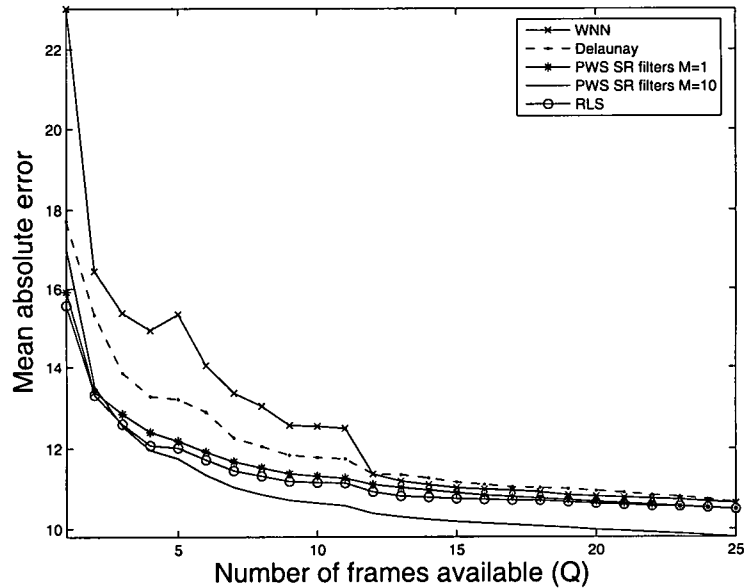


Figure 8.7: Mean absolute error (MAE) as a function of the number of input frames, Q , for various SR methods applied to the simulated image sequences.

that yields the minimum error for each sequence. For the RLS method, we select the tuning parameter, λ , in [1] that yields the minimum error for each sequence.

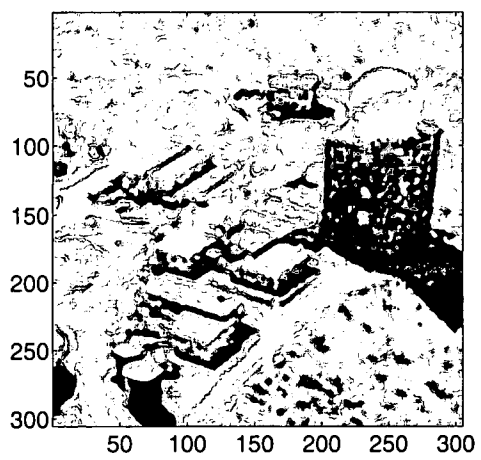
Note that all the methods yield lower error as the number of unique frames is increased. Also, note that the PWS SR filter with $M = 10$ provides the lowest error for $Q \geq 3$. It is observed from Fig. 8.7 that with $Q < 3$ the RLS method and PWS SR method with $M = 1$ outperform the $M = 10$ PWS SR method. We believe that this occurs because the VQ partitioning is based on too few samples to effectively differentiate key structural information in the HR grid. However, as the number of unique frames increases the PWS based SR approach is better able to identify and exploit the local intensity characteristics through the VQ partitioning process. The

error plots were generated using the script file *This plot was generated using the script file `runsimsuperres.m`*

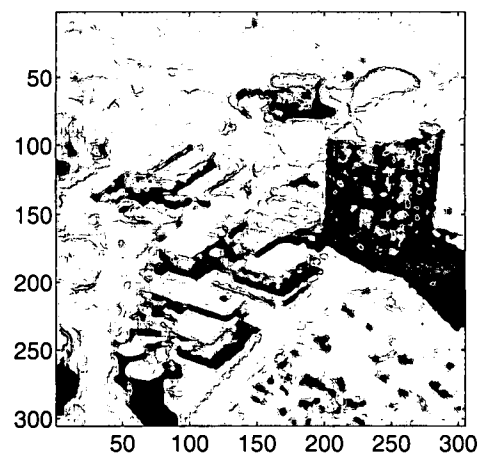
To subjectively evaluate the performance of the SR methods, let us consider the image sequence with $Q = 5$ simulated LR frames. The partially populated HR grid is shown in Fig. 6.4. The HR image reconstructed using PWS SR filters with $M = 1$ is shown in Fig. 8.8(a). We believe this shows significant improvement in resolution over the LR image shown in 8.1(b). The HR image estimated by applying PWS SR filters with $M = 10$ is shown in Fig. 8.8(b). Note that with 10 partitions the HR image estimate has reduced noise compared to the image shown in Fig. 8.8(a). The outputs of the RLS algorithm, the WNN SR algorithm, and the Delaunay triangulation algorithm are shown in Figs. 8.8(c), (d) and (e), respectively. For reference, a single frame bicubic interpolated image is shown in Fig. 8.8(f). The images shown in Fig. 8.8(a) and (b) were generated using Matlab functions *blockprocess.m* and *partitionblockprocessforovrlp.m* respectively. The images shown in Fig. 8.8(d) and (e) were generated using the Matlab functions *nonuniforminterpolation.m* and *triangtrans.m* respectively.

8.2 Infrared Data

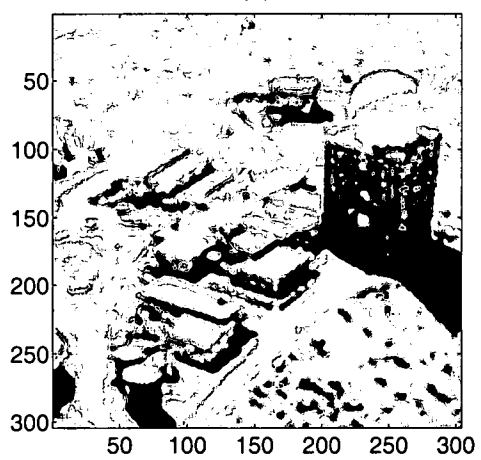
In this section we present the results obtained by applying PWS SR filters to two real infrared video sequences. One sequence has global translational motion and the other contains translational and rotational motion. These data were provided by the Sensors Technology Branch at the Air Force Research Laboratories, Wright Patterson Air Force Base. The video sequences were acquired using a forward looking infrared (FLIR) camera equipped with a 128×128 Amber AE-4128 infrared FPA. The FPA is



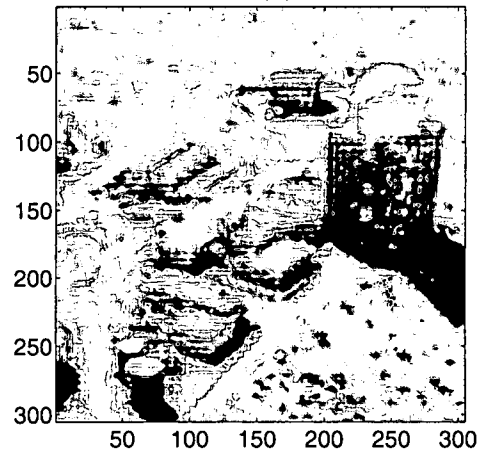
(a)



(b)



(c)



(d)

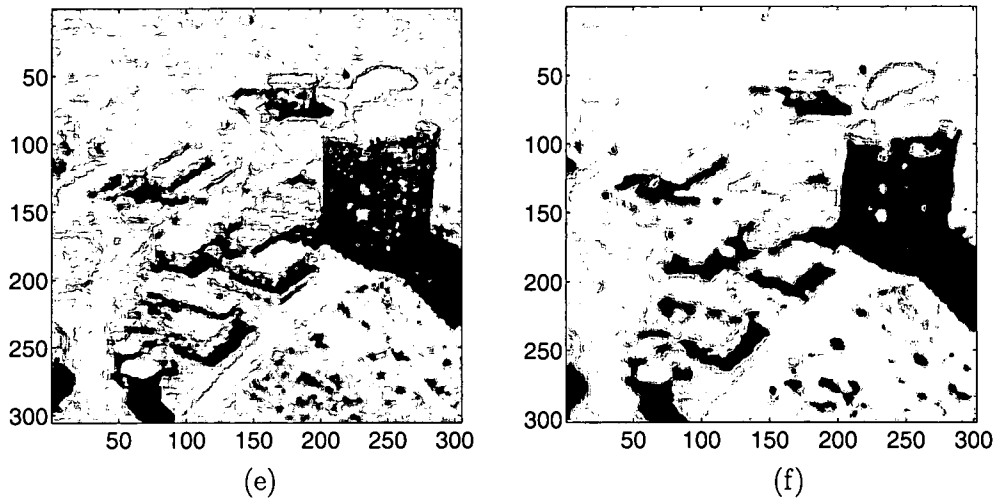


Figure 8.8: HR image estimate for several SR algorithms using $Q = 5$ frames. (a) PWS SR filter with $M = 1$, $N_1 = N_2 = 15$, and $K_1 = K_2 = 5$. (b) PWS SR filters with $M = 10$, $N_1 = N_2 = 15$, $P_1 = P_2 = 7$, and $K_1 = K_2 = 5$. (c) RLS image reconstruction technique (d) WNN method (e) Delaunay triangulation (f) bicubic interpolation of Frame 1.

composed of Indium-Antimonide (InSb) detectors with a response in the $3\mu\text{m} - 5\mu\text{m}$ wavelength band. This system has square detectors of size $a = b = 0.040$ mm. The FLIR camera is equipped with a lens of aperture opening 100mm in diameter and an f-number of $f/3$, yielding a cut-off frequency of 83.3 cycles/mm. The detector spacing on the Amber FPA is 0.050 mm, yielding a sampling frequency of 20 cycles/mm in both directions. Thus, the effective sampling rate must be increased by a factor of 8.33 to eliminate aliasing entirely for an arbitrary scene. This would require that we select $L_1 = L_2 \geq 9$. In practice, we find that good results can be obtained with $L_1 = L_2 = 5$.

8.2.1 Translational Motion

The translational FLIR video sequence consists of 100 LR frames. Each frame has been cropped down to a size of 64×64 from the original 128×128 data. The global translational motion between the frames is achieved by panning the infrared imager during image acquisition. Figure 8.9 shows a LR frame from the sequence. The scene contains trees surrounding a house located by the side of a road. The shift estimates, computed with the gradient based registration technique, are shown in Fig 8.10.

We process the LR video sequence to produce an HR video sequence with $L_1 = L_2 = 5$ using a sliding group of $Q = 15$ frames. For each method, we process only the portion of the HR grids where all of the Q input frames overlap spatially. For the PWS SR methods we use $N_1 = N_2 = 15$, $K_1 = K_2 = 5$, and $P_1 = P_2 = 7$. The first output HR frame, obtained using various SR methods, is shown in Fig. 8.11. The PWS SR method outputs with $M = 1$ and $M = 10$ are shown in Figs. 8.11(a) and (b), respectively. The results obtained with the RLS image reconstruction

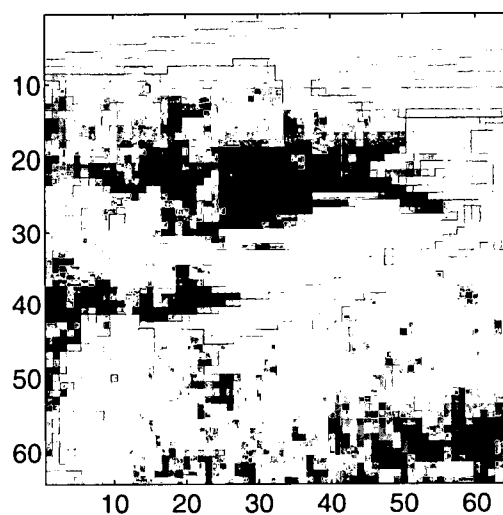


Figure 8.9: LR frame 1 of infrared image sequence data with translational motion.

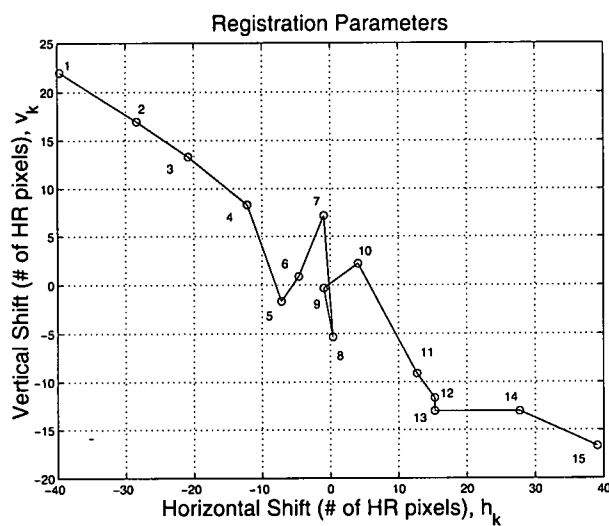


Figure 8.10: Registration parameters in HR pixels for the $Q = 15$ frame infrared image sequence with translational motion.

algorithm (4 iterations and $\lambda = 0.005$), WNN SR algorithm (NSR = 0.03), and the Delaunay triangulation followed by Wiener filtering (NSR = 0.03) are shown in Figs. 8.11(c), 8.11(d), and 8.11(e), respectively. A single frame interpolated using bicubic interpolation is shown in Fig. 8.11(f) as a reference. We believe that the multiframe approaches provide significant improvement in resolution over the single frame and the PWS SR method appears to provide the sharpest images.

Figure. 8.11(a) shows a portion of the reconstructed HR grid. To achieve the full size of the HR image, the reconstruction is carried out in two steps. The reason for performing the reconstruction in two steps is that we wish to take advantage of the repetitive structure of the HR grid, which helps in reconstructing a larger portion of the grid with a single weight matrix. We first identify the region in the HR grid where all the LR frames overlap spatially and then apply PWS SR with $M = 1$, $N_1 = N_2 = 15$ and $K_1 = K_2 = 5$ to reconstruct this region. This is followed by processing the borders of the HR grid separately using $M = 1$, $N_1 = N_2 = 15$ and $K_1 = K_2 = 15$. Note that we are using a larger size estimation window to expedite the reconstruction process along the borders of the HR grid. The full size HR image after border processing is shown in Fig. 8.12.

The proposed algorithm is implemented in MATLAB on a pentium IV 2.8GHz PC with 2GB RAM. The time taken to generate the HR images along with a flop count per HR pixel are listed in Table 8.1. The flop count is provided using the performance application programming interface API (PAPI) tool for MATLAB. These numbers in Table 8.1 do not include the image registration process or formation of the partially populated HR grid. In the case of PWS SR filters with $M = 1$, the run time shown in Table 8.1 is 0.187 seconds. Note, however, that approximately 0.11 seconds of

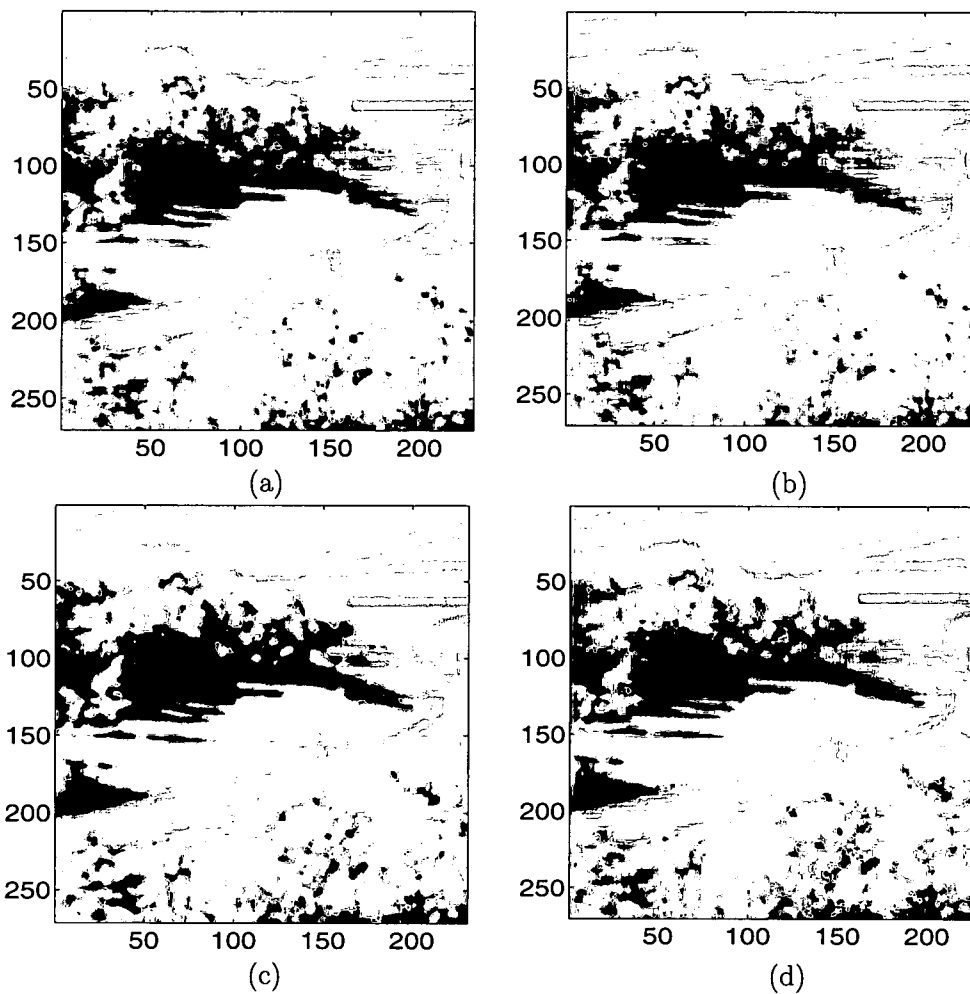


Figure 8.11: SR images generated from $Q = 15$ infrared image frames with translational motion. (a) PWS SR filters with $M = 1$, $N_1 = N_2 = 15$ and $K_1 = K_2 = 5$, (b) PWS SR filters with $M = 10$, $N_1 = N_2 = 15$, $P_1 = P_2 = 7$ and $K_1 = K_2 = 5$, (c) RLS image reconstruction algorithm, (d) WNN method, (e) Delaunay triangulation, (f) single frame bicubic interpolation.

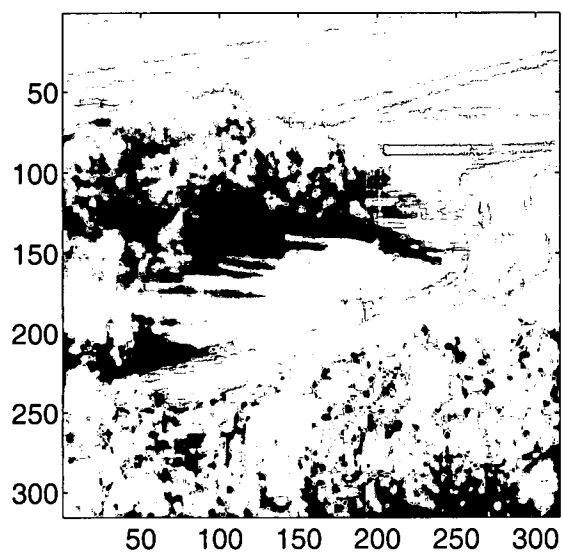
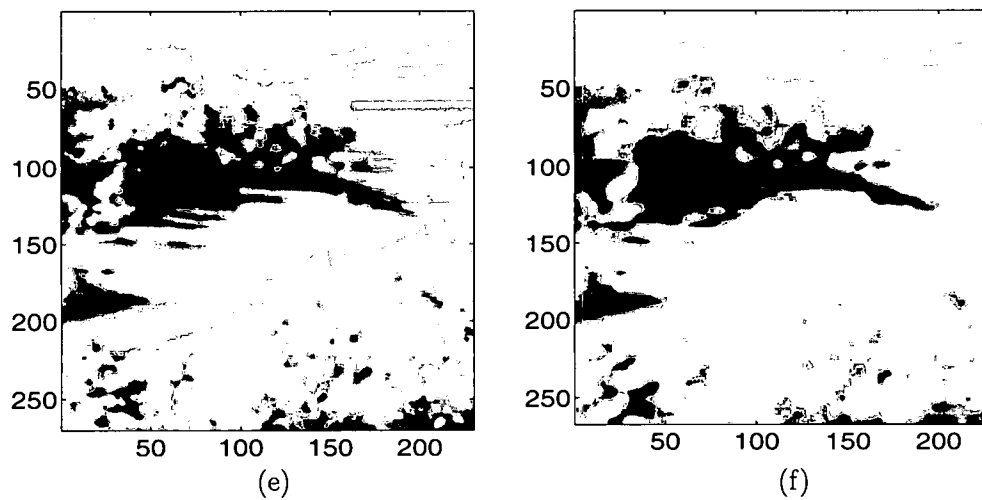


Figure 8.12: Full size HR image after border processing using PWS SR filters.

Table 8.1: Run time and flops per output HR pixel for FLIR image results with translational motion.

Algorithm	PAPI Flop Count	Matlab Run Time
PWS SR filters $M=1$	305	0.187s
PWS SR filters $M=10$	541	0.202s
RLS image reconstruction	19537	14.20s
WNN (IIR Wiener filtering)	1079	0.296s
Delaunay triangulation	2630	4.53s
Single frame bicubic interpolation	233	0.297s

this is consumed by memory management in forming the observation vectors and the filtering operation takes 0.07s. For PWS SR filters with $M = 10$, the run time is 0.202s, of which 0.015s is used in precomputing the filter weights. There is not much of overhead in using $M = 10$ partitions. It is observed that PWS SR filters take less time to form HR image estimate using multiple frames compared to the single frame bicubic interpolation. The run time and flop count for PWS SR filters is lower than the WNN SR method and the RLS HR image reconstruction technique. Note that the registration takes approximately 0.03 seconds per frame and populating the HR grid takes approximately 0.031s.

8.2.2 Translation and Rotation

The second infrared video sequence consists of $Q = 20$ LR video frames each of size 64×64 that include global translational and rotational motion. One typical LR frame from the sequence is shown Fig. 8.13. The scene contains small power boats and trailers with a fence in the foreground. The registration parameters estimated for the 20 frames are shown in Fig. 8.14.

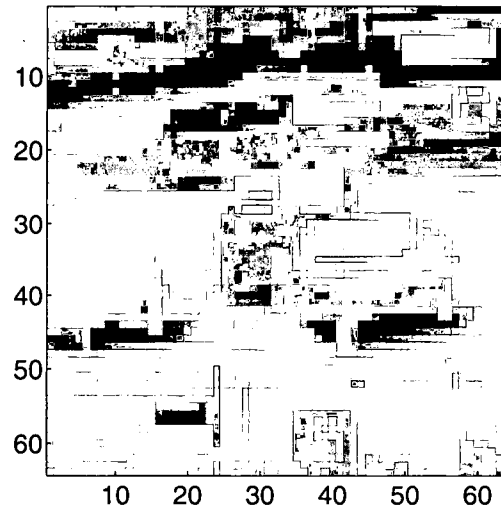


Figure 8.13: LR frame 1 of infrared image sequence data with translational and rotational motion.

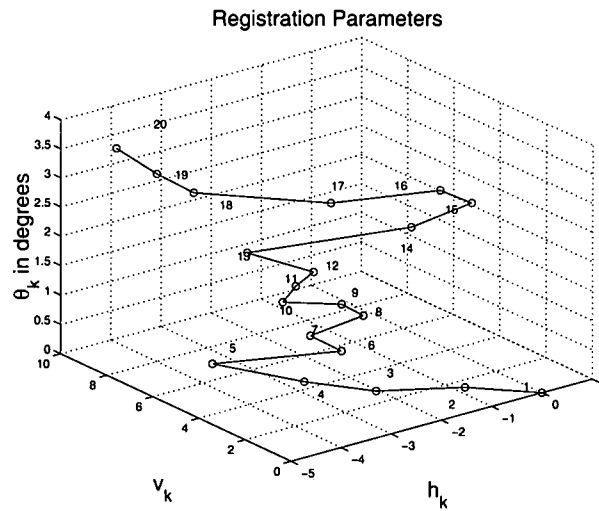


Figure 8.14: Registration parameters in HR pixels for the $Q = 20$ frame infrared image sequence with translational and rotational motion.

Table 8.2: Run time and flops per output HR pixel for FLIR results with translational and rotational motion.

Algorithm	PAPI Flop Count	Matlab Run Time
PWS SR filters $M=1$	70211	9.7s
PWS SR filters $M=10$	70639	10.6s
RLS image reconstruction	25526	25s
Delaunay triangulation	3033	6.12s
Single frame bicubic interpolation	233	0.297s

We process the LR video sequence to produce an HR image using the same parameters as for translation, except here $Q = 20$. The output images, obtained using several SR methods, are shown in Fig. 8.15. The PWS SR method outputs with $M = 1$ and $M = 10$ are shown in Figs. 8.15(a) and (b), respectively. These images were generated using Matlab functions *blockprocess.m* and *partitionblockprocess.m* respectively. The output obtained with the RLS image reconstruction algorithm (4 iterations and $\lambda = 0.005$) is shown in 8.15(c). A single frame interpolated using bicubic interpolation is shown in Fig. 8.15(d) as a reference.

Table 8.2 provides the MATLAB run times and the flops per output HR pixel for the rotation sequence. Note that due to the rotational motion in the LR frames, the repetitive population structure is no longer preserved over the entire region of the HR grid. This increases the computational complexity of the PWS SR algorithm.

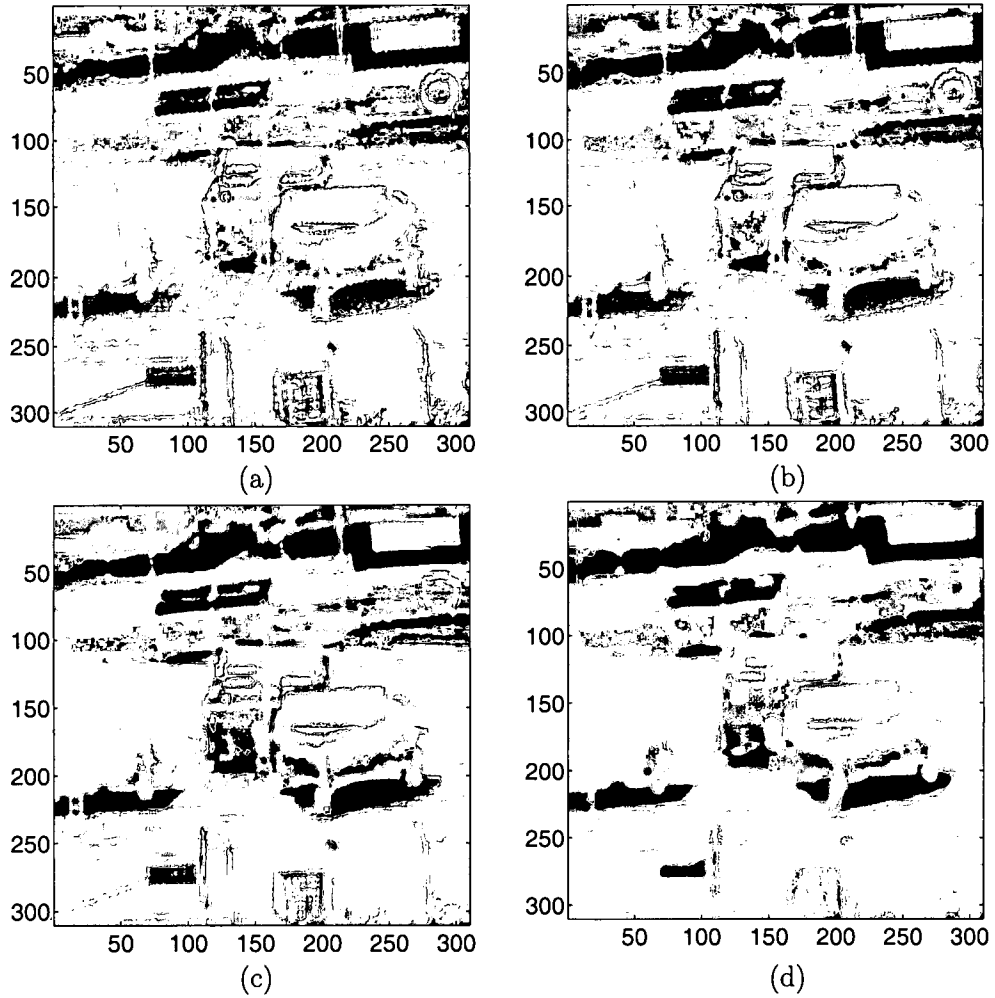


Figure 8.15: SR images generated from $Q = 20$ infrared image frames with translational and rotational motion. (a) PWS SR filter with $M = 1$, $N_1 = N_2 = 15$ and $K_1 = K_2 = 5$ (b) PWS SR filter with $M = 10$, $N_1 = N_2 = 15$, $P_1 = P_2 = 7$ and $K_1 = K_2 = 5$ (c) RLS image reconstruction algorithm (d) bicubic interpolation of frame 1.

CHAPTER 9

CONCLUSIONS

We have developed a new scene-based NUC technique that treats the aggregate nonuniformity resulting from the readout electronics and the individual detectors in two separate stages. It has been shown that a useful correction can be obtained using few frames. The algorithm was tested on both simulated and real infrared image sequences. The size of the median filter required to provide the preliminary scene estimate is dependent on the sensor. The proposed algorithm does not treat low frequency nonuniformity noise.

A region-based WNN nonuniform interpolation technique is employed to reconstruct the borders of the HR grid. This technique is very useful when there is large motion between the LR frames.

We have also presented a novel PWS filter training and implementation methodology for SR processing of video. In our approach, the bulk of the filter optimization is done prior to image acquisition. This makes it practical for video processing. The proposed approach requires that the scene remains approximately constant with rigid motion. For the case of translational motion, minimal computations are required per frame to obtain all of the needed filter weights. Furthermore, the proposed algorithm lends itself easily to a parallel implementation. The computational complexity study

presented here shows that, for translational motion, the proposed method has a computational complexity similar to that of the very simple WNN algorithm [36, 40]. The implementation time for PWS SR filters is almost the same as that of the single frame bicubic interpolation technique. For arbitrary motion, the complexity of the PWS method increases, but still produced run times that are less than half of that of the RLS image reconstruction method [1] in our implementations.

In our quantitative experimental results (translational motion), the PWS SR method with $M = 10$ outperformed all of the benchmark methods for $Q \geq 3$. These quantitative results also clearly show the benefit of the partitioning process used in the PWS SR algorithm. In particular, the PWS SR method using $M = 10$ partitions outperformed the similar processing with no partitioning (i.e., $M = 1$). Furthermore, based on subjective evaluation, we believe that the PWS SR method generally produced the sharpest images compared with the benchmark techniques. The performance of the PWS SR can be improved by forming the HR grid without quantizing the shifts of the LR frames. The PWS based SR technique, like other interpolation based SR techniques relies on the observed data and does not provide end to end optimality. Possible future work in this area is the real time implementation of PWS SR filters.

In summary, we believe that the good performance of the proposed PWS SR method, combined with a relatively low computational complexity and highly parallel structure, makes it attractive for real-time SR enhancement of video. Furthermore, in addition to SR video reconstruction, we believe that the proposed PWS methodology is a potentially useful tool for addressing a wide range of other nonuniform interpolation problems.

APPENDIX A

MATLAB SOURCE CODE FOR NUC ALGORITHM

Main program for NUC of simulated data

runsimnuc.m

Run file to generate noisy image with artificial gain and bias The channel noise and detector noise is simulated and added to the frames. This code also corrects the channel and detector non uniformity separately using readout architecture and RLS technique respectively. This code also plots the MAE and SNR. Calls functions scan, detectornu, addchannnoise, framechanncorr, trimmean, med2d and recurlsstda

Date: 07/25/04

```
close all; clear all; clc;
%load true image
load kenaerial1;
Crepl=kenaaerial1; %size of frames along x and y
sz=256;
%number of frames
nfrm=100;
%generate frames
obs=scan(Crepl,sz,nfrm);
%size of data [sy,sx,N]=size(obs);
%g and b are the artificially generated gain and bias
%coefficients
g=randn(size(obs(:,1))) * 0.02 + 1;
b=randn(size(obs(:,1))) * 0.5 * std(obs(:));
%allocate memory
med1=zeros(size(obs)); med2=zeros(size(obs));
med3=zeros(size(obs)); med4=zeros(size(obs));
out1=zeros(size(obs)); out2=zeros(size(obs));
out3=zeros(size(obs)); outstd1=zeros(size(obs));
outstd2=zeros(size(obs)); outstd3=zeros(size(obs)); C=obs; nobis=zeros(size(obs));
```

```

corrfrm=zeros(size(obs));
%generate the noisy frames(individual detector noise)
detn=detectornu(obs,g,b);
%add channel noise
nobs=addchannoise(detn);
%indices for extracting channel outputs
in1=[1:1:sy]; in2=[1:4:sx]; in3=[2:4:sx];
in4=[3:4:sx]; in5=[4:4:sx];
%alpha trim factor to compute statistics
alpha=3;
%Apply channel nuc
corrfrm=framechanncorr(nobs,in1,in2,in3,in4,in5,alpha);
%2M1+1 is the length median filter
%three different median
filters M1=2; M2=4; M3=6;
%perform median filtering
for i=1:N
fk=corrfrm(:,i);
disp(sprintf('filtering frame %d',i));
%median filtering on channel corrected frames
med1(:,i)=med2d(softpad(fk,M1,M1,M1,M1),2*M1+1,2*M1+1);
med2(:,i)=med2d(softpad(fk,M2,M2,M2,M2),2*M2+1,2*M2+1);
med3(:,i)=med2d(softpad(fk,M3,M3,M3,M3),2*M3+1,2*M3+1);
%perform median filtering on frame without channel correction
med4(:,i)=med2d(softpad(nobs(:,i),M1,M1,M1,M1),2*M1+1,2*M1+1);
end
figure;
%apply RLS based NUC to correct detector nonuniformity after channel NUC
%median filter size is 5x5
[Gr1s1,Brls1,out1,outstd1,errrls1,errstd1,snrrls1,snrstd1]=recurlssda(...
corrfrm,nobs, med1,obs,max(nobs(:)),1,M3);
%apply RLS based NUC to correct detector nonuniformity without channel NUC
%median filter size is 5x5
figure;
[Gr1s11,Brls11,out11,outstd11,errrls11,errstd11,snrrls11,...
snrstd11]=recurlssda(nobs,nobs, med4,obs,max(nobs(:)),1,M3);
figure;
% apply RLS based NUC to correct detector nonuniformity after channel NUC
%median filter size is 9x9
[Gr1s2,Brls2,out2,outstd2,errrls2,errstd2,snrrls2,snrstd2]=recurlssda(...
corrfrm,nobs,med2, obs,max(nobs(:)),1,M3);
figure;
% apply RLS based NUC to correct detector nonuniformity after channel NUC
% median filter size is 13x13
[Gr1s3,Brls3,out3,outstd3,errrls3,errstd3,snrrls3,snrstd3]=recurlssda(...

```

```

corrfrm,nobs,med3, obs,max(nobs(:),1,M3);
%plot error (MAE)
st=3; frame=[st:1:N];
h1=figure;
plot(frame,errrls1(st:N),'r-.',frame,errrls2(st:N),'b-*',frame,errrls3(st:N),'c-o');
set(get(h1,'CurrentAxes'),'FontSize',10);
title('error Vs frame');
xlabel('Number of frames','FontSize',18);
ylabel('Mean Absolute Error','FontSize',18);
legend('5x5 Median Filter','9x9 Median Filter','13x13 Median',1);
% print error.ps
%plot SNR
h34=figure;
plot(frame,snrrls1(st:N),'r-.',frame,snrrls2(st:N),'b-*',frame,snrrls3(st:N),'c-o');
set(get(h34,'CurrentAxes'),'FontSize',10);
title('SNR Vs frame');
xlabel('Number of Frames','FontSize',18);
ylabel('Signal to Noise Ratio','FontSize',18);
legend('5x5 Median Filter','9x9 Median Filter','13x13 Median Filter',2);
% print snr.ps
%Display images
h3=figure; imstd(Crepl);
set(get(h3,'CurrentAxes'),'FontSize',20);
%print tvi.ps
h4=figure; imstd(obs(:,:,N))
set(get(h4,'CurrentAxes'),'FontSize',20);
%print raw.ps
h5=figure; imstd(detn(:,:,N));
set(get(h5,'CurrentAxes'),'FontSize',20);
%print detn.ps;
h6=figure; imstd(nobs(:,:,N));
set(get(h6,'CurrentAxes'),'FontSize',20);
title('after channel noise');
% print noisyfrm.ps
h7=figure; imstd(corrfrm(:,:,N));
set(get(h7,'CurrentAxes'),'FontSize',20);
title('after channel correction');
%print chanco.ps
h8=figure; imstd(med1(:,:,N))
set(get(h8,'CurrentAxes'),'FontSize',20);
title('Median Filtered (Preliminary Scene Estimate) after readout correction')
% print med5.ps
h9=figure; imstd(out1(:,:,N))
set(get(h9,'CurrentAxes'),'FontSize',20);
title('Corrected Frame Using RLS Parameters (100 Frames) after channel correction')

```

```
%print rls5.ps  
h10=figure; imstd(out11(:, :, N))  
set(get(h10,'CurrentAxes'),'FontSize',20);  
title('Corrected Frame Using RLS Parameters (100 Frames) after channel correction') %print  
rls5nch.ps
```

scan.m

```
function out=scan(in,sz,nframe);
```

Function to generate a sequence of frames from given image by scanning the image diagonally.

in is the input image

sz is the size of each frame along x and y (all frames must be of uniform size)

nframe is the number of frames

Author: Balaji Narayanan

Date: 01/09/04

```
%size info
[sy,sx]=size(in);
%minimum
sm=min([sy,sx]);
%maximum number of frames available
maxfrm=sm-sz+1;
%check for number of frames
if(nframe>maxfrm)
disp('enter fewer number of frames');
else
%step size to jump
step=floor(maxfrm/nframe);
%generate each frame
for i=1:nframe
temp=(i-1)*step;
out(:, :, i)=in(1+temp:sz+temp,1+temp:sz+temp);
end
end
```


detectornu.m

```
function nobs=detectornu(obs,g,b);  
%function nobs=calibdeg(obs,g,b);  
This function adds detector nonuniformity to the true frames using the artificial gains and  
biases.  
Author: Balaji Narayanan  
Date: 01/09/04  
  
%size info [sy,sx,N]=size(obs);  
%multiply each frame by gain and add bias  
for k=1:N  
nobs(:,k)=obs(:,k).*g+b;  
end
```

addchannoise.m

```
function out=addchannoise(in);
% function out=addchannoise(in);
This functions adds channel noise to the input image. The read out pattern used here is
such that every 4th column has same gain and bias.
out is the output after adding noise.
in is the input image.
Author: Balaji Narayanan
Date: 01/19/04

%size of the input image
[sy,sx,n]=size(in);
%create indices
ind1=[1:4:sx]; ind2=[2:4:sx];
ind3=[3:4:sx]; ind4=[4:4:sx];
%generate the gain to simulate %channel noise at different levels
g1=(randn(1,1)*0.15+1.25).*ones(sy,length(ind1),n);
g2=(randn(1,1)*0.25+1.5).*ones(sy,length(ind2),n);
g3=(randn(1,1)*0.35+1.75).*ones(sy,length(ind3),n);
g4=(randn(1,1)*0.45+2).*ones(sy,length(ind4),n);
%generate the bias to simulate channel noise at
% different levels
b1=(randn(1,1)*0.55*std(in(:))).*ones(sy,length(ind1),n);
b2=(randn(1,1)*0.65*std(in(:))).*ones(sy,length(ind2),n);
b3=(randn(1,1)*0.70*std(in(:))).*ones(sy,length(ind3),n);
b4=(randn(1,1)*0.85*std(in(:))).*ones(sy,length(ind4),n);
%generate the noisy output image
in(:,ind1,1:n)=in(:,ind1,1:n).*g1+b1; in(:,ind2,1:n)=in(:,ind2,1:n).*g2+b2;
in(:,ind3,1:n)=in(:,ind3,1:n).*g3+b3; in(:,ind4,1:n)=in(:,ind4,1:n).*g4+b4;
out=in;
```

trimmean.m

```
function [tmean,tvar]=trimmean(data,alpha);
% function [tmean,tvar]=trimmean(data,alpha);
This function evaluates the trimmed mean and variance. The trim factor is used to eliminate outliers from the mean and variance calculation.
data is input vector
Alpha is the trim factor in percentage.
Author: Balaji Narayanan
Date : 01/09/04

%length of vector
len=length(data);
% number of data elements to eliminate
tlen=round(alpha*len/100);
%sort the data
sdata=sort(data);
% index
st=floor(tlen); en=tlen-st;
%compute mean and variance
tmean=mean(sdata(st+1:len-en));
tvar=var(sdata(st+1:len-en),1);
```

framechanncorr.m

%fun correctfrm=framechanncorr(obs,in1,in2,in3,in4,in5,alpha);
This function performs the channel level nonuniformity correction. Normlizes the pixels
belonging to each group amplifier.

Author: Balaji Narayanan

date: 11/11/03

```
%initialize variables
[sy,sx,N]=size(obs);
meansub1=0; meansub2=0;
meansub3=0; meansub4=0;
varsub1=0; varsub2=0;
varsub3=0; varsub4=0;
%start processing each
frame for k=1:N
disp(sprintf('processing frame %d',k));
%extract the channel groups from each frame
framsub1=obs(in1,in2,k);
framsub2=obs(in1,in3,k);
framsub3=obs(in1,in4,k);
framsub4=obs(in1,in5,k);
%compute the alpha trimmed mean and variance for each group
[tmean1,tvar1]=trimmean(frams1(:),alpha);
[tmean2,tvar2]=trimmean(frams2(:),alpha);
[tmean3,tvar3]=trimmean(frams3(:),alpha);
[tmean4,tvar4]=trimmean(frams4(:),alpha);
%compute the mean over frames(time)
meansub1=(meansub1*(k-1)+tmean1)/k;
meansub2=(meansub2*(k-1)+tmean2)/k;
meansub3=(meansub3*(k-1)+tmean3)/k;
meansub4=(meansub4*(k-1)+tmean4)/k;
%compute the variance over frames(time)
varsub1=((k-1)*varsub1+tvar1)/k;
varsub2=((k-1)*varsub2+tvar2)/k;
varsub3=((k-1)*varsub3+tvar3)/k;
varsub4=((k-1)*varsub4+tvar4)/k;
%normalize the four groups
cfmnorm1=(framsub1-meansub1)./(sqrt(varsub1));
cfmnorm2=(framsub2-meansub2)./(sqrt(varsub2));
cfmnorm3=(framsub3-meansub3)./(sqrt(varsub3));
cfmnorm4=(framsub4-meansub4)./(sqrt(varsub4));
%compute the average mean and standard deviation over the
%four groups
me=mean([meansub1,meansub2,meansub3,meansub4]);
st=sqrt([varsub1,varsub2,varsub3,varsub4]);
```

```

%insert the groups
normframe(in1,in2)=cfmnorm1;
normframe(in1,in3)=cfmnorm2;
normframe(in1,in4)=cfmnorm3;
normframe(in1,in5)=cfmnorm4;
% normalise the current frame with common mean
% standard deviation
correctfrm(:,k)=normframe.*st+me;
end

```

med2d.m

```
function out=med2d(in,wsx,wsy);  
out=med2d(in,wsx,wsy)  
2D Median filter of size wsx x wsy  
out Filtered image  
in Input image  
wsx Window size in x  
wsy Window size in y  
Author: Dr. Russell C. Hardie  
University of Dayton  
Date: 4/17/98  
  
[sy,sx]=size(in); bx=(wsx-1)/2; by=(wsy-1)/2;  
A = im2col(in,[wsy,wsx],'sliding'); A2= median(A);  
out=reshape(A2,sy-2*by,sx-2*bx);
```

recurlstda.m

```
[GrIs,Brls,out,outstd,errrls,errstd,snrrls,snrstd]=...
```

```
recurlstda(obs,obs1,med,C,sat,lam,ind);
```

Nonuniformity parameter estimator using simulated data. The artificially degraded frames are corrected using RLS and global STD methods. The simulated data is fed into a recursive least squares algorithm to update the Gain and Bias parameters. A "forgetting factor" is used to allow for drift in these parameters. Only outputs the Final Gain and Bias,error,SNR.

obs channel NU corrected frames

obs1 3D array of observed frames with NU

med median filtered frames

C true frames used for computing errors

sat Saturation value of the detector.Used to clip out-of-range pixels in the output.

lam forgetting factor 0-1. 1 is growing window RLS (no forgetting).

ind index from where the error is computed

Original Author: Russell C. Hardie

Created from comprlsstdmed.m on 9/3/00

Modified by Balaji Narayanan

Date: 11/11/03

```
%-----%
```

```
% Initialize loop variables %
```

```
%-----%
```

```
[sy,sx,N]=size(obs);
```

```
out=zeros(sy,sx); out2=zeros(sy,sx);
```

```
GrIs=ones(sy,sx); Brls=zeros(sy,sx);
```

```
Gstd=ones(sy,sx); Bstd=zeros(sy,sx);
```

```
meanobs=zeros(sy,sx); stdobs=zeros(sy,sx);
```

```
del=.01; PP=ones(sy,sx,4); PP(:,:,1)=(1/del)*PP(:,:,1);
```

```
PP(:,:,2)=zeros(sy,sx); PP(:,:,3)=zeros(sy,sx);
```

```
PP(:,:,4)=(1/del)*PP(:,:,4);
```

```
S1=zeros(N,2); S2=zeros(N,2);
```

```
for k=1:N
```

```
disp(sprintf('Processing frame %d',k))
```

```
%compute signal power for each frame
```

```
sigpow=C(ind+1:end-ind,ind+1:end-ind,k);
```

```
sigpow=var(sigpow(:));
```

```
fk=squeeze(obs(:,:,k));
```

```
fk1=squeeze(obs1(:,:,k));
```

```
%-----%
```

```
% Recursive LS %
```

```
%-----%
```

```
Pred=GrIs.*med(:,:,k)+Brls;
```

```
Alpha=fk-Pred;
```

```
Z1=PP(:,:,1).*med(:,:,k)+PP(:,:,3);
```

```

Z2=PP(:,:,2).*med(:,:,k)+PP(:,:,4);
gc=1./(lam+med(:,:,k).*Z1+Z2);
G1=gc.*Z1;
G2=gc.*Z2;
Gr1s=Gr1s+Alpha.*G1;
Br1s=Br1s+Alpha.*G2;
PP(:,:,1)=(1/lam)*(PP(:,:,1)-G1.*Z1);
PP(:,:,2)=(1/lam)*(PP(:,:,2)-G2.*Z1);
PP(:,:,3)=(1/lam)*(PP(:,:,3)-G1.*Z2);
PP(:,:,4)=(1/lam)*(PP(:,:,4)-G2.*Z2);
%-----%
% Now STD METHOD %
%-----%
meanobs=(fk1+(k-1)*meanobs)/k;
stdobs=(abs(fk1-meanobs)+(k-1)*stdobs)/k;
Izero=find(stdobs==0);
stdobs2=stdobs;
stdobs2(Izero)=ones(size(Izero));
Gstd=stdobs2;
Bstd=meanobs;
%-----%
% Correct current frame %
%-----%
% For STD
out2=(fk1-Bstd)./Gstd;
mfk=mean(fk1(:));
sfk=std(fk1(:));
m2=mean(out2(:));
s2=std(out2(:));
outstd(:,:,k)=clip(((out2)/s2).*sfk+mfk,0,sat);
% For RLS
out(:,:,k)=clip((obs(:,:,k)-Br1s)./Gr1s,0,sat);
%RLS
ou1=out(ind+1:end-ind,ind+1:end-ind,k);
ou2=C(ind+1:end-ind,ind+1:end-ind,k);
rlsnopow=ou1-ou2;
errrls(k)=mean2(abs(rlsnopow));
rlsnopow=var(rlsnopow(:));
%SNR for RLS
snrrls(k)=sigpow/rlsnopow;
%compute error for STD
stdnopow=((outstd(ind+1:end-ind,ind+1:end-ind,k)-C(ind+1:end-ind,
ind+1:end-ind,k)));
errstd(k)=mean2(abs(stdnopow));
stdnopow=var(stdnopow(:));

```



```
%SNR for STD
snrstd(k)=sigpow/stdnopow;
% Display results
imstd([C(:,k),obs(:,k);out(:,k),outstd(:,k)]);
drawnow
end
```

APPENDIX B

MATLAB SOURCE CODE FOR REGION-BASED WNN NONUNIFORM INTERPOLATION

Main program for border processing

runfastbordersuperresolution.m

Run file for border processing using region based WNN non-uniform interpolation

```
close all; clear all; clc;
%load the data
tot=100; fid=fopen('test100','r+','b');
data=fread(fid,64*64*tot,'float');
data=reshape(data,64,64,tot);
%initialize variables
%low res pixels
lrsx=64; lrsy=64; LX=4; LY=4;
%high res pixel
hy=lrsy*LY; hx=lrsx*LX; maxNN=4;%maximum neighbours
minNN=16;%minimum neighbors and step=1 for all 16 frames
%total frames fed in
totinframes=17;
%number of frames used for reconstruction
numframes=16;
%——precompute the wiener filter response——% %points spread function
psf=ones(LY,LX)/(LX*LY);
%noise to signal ratio
nsr=0.1;
wbuff=20; % border buffer to minimize ringing at borders
% wbuff=0 means ringing might be bad at edges
% so one has to cut out the center of the image.
% a wbuff of 10-20 usually work well but adds to the fft sizes.
```

```

%weiner filter response
H=fft2(psf,hy+2*wbuff,hx+2*wbuff);
absH2=(abs(H).^2); HW=conj(H)./(absH2+nsr);
in=data(:,1:totinframes); tic
[z,zw,cord,S]=fastbordersuperresolution(in,lrsx,lrsy,LX,LY,...
numframes,maxNN,minNN,1,HW); toc
%high res image after filtering
zw1=zeros(size(z));
zw1(cord(1,1):cord(2,1),cord(1,2):cord(2,2),1)=zw(cord(1,1):cord(2,1),...
cord(1,2):cord(2,2),1);
%display image
figure; imstd1(z(:, :, 1)); figure imstd1(zw1(:, :, 1));

```

fastbordersuperresolution.m

```
function [out,outwiener,cord,shifts]=fastbordersuperresolution(lowobs,lrsx,...
lrsy,LX,LY,numframes,maxNN,minNN,step,HW);
Step by step reconstruction of high resolution image from multiple low resolution frames
with translational shift. Takes multiple low res frames (lowobs) and estimates the proper
values on the high resolution grid using step by step reconstruction. The high res grid is
centered about the **AVERAGE** x,y position of the input frames.
out high resolution images
outwiener wiener filtered high resolution images
cord coordinates of the processed region in HR grid
shifts translational shifts for each frame measures in
low resolution pixel spacings S=[sx1,sy1; sx2,sy2; ...];
lowobs low resolution images lowobs is matrix of size
lrsy X lrsx X nframes
lrsx Horizontal low res image size
lrsy Vertical low res image size
LX The desired upsample factor in X
LY The desired upsample factor in Y
numframes number of frames used to reconstruct one high res frame
maxNN,minNN maximum and minimum number of neighbors used to
weight each pixel (1 - number of frames). 1=nearest neighbor.
HW Wiener filter frequency response
Author: Balaji Narayanan
date 10/26/04

%size info
[ly,lx,totinframes]=size(lowobs);
%hig res size
hy=LY*lrsy; hx=LX*lrsx;
%number of frames required based on the number
%of neighbors used for interpolation
nfrms=numframes-minNN;
if(step > nfrms)
step=1;
steparray=[0:step:nfrms];
else
steparray=[0:step:nfrms];
end
nfrms=length(steparray);
%total number of output frames
outnumframes=totinframes-numframes+1;
%initialize the output z and zw
out=zeros(hy,hx,outnumframes);
outwiener=zeros(hy,hx,outnumframes);
%compute the shifts by registering
```

```

shifts1 = zeros( numframes,2 );
for frameidx = 2 : numframes-1
    incrementals1 = registershift(lowobs( :, :, frameidx - 1 ),...
    lowobs( :, :, frameidx ), .05, round(0.1*lrsl));
    shifts1( frameidx, : ) = shifts1( frameidx - 1, : ) + incrementals1';
end
for frameidx = numframes : totinframes
    % Register the current frame
    incrementals1 = registershift( lowobs( :, :, frameidx - 1 ),... lowobs( :, :, frameidx ),.05,
    round( .1 * lrsl ) );
    shifts1(frameidx, : ) = shifts1( frameidx-1, : ) + incrementals1';
    %select the required frames as given by numframes
    in=lowobs(:, :,frameidx-numframes+1:frameidx);
    %compute the shifts of the frames wrt the average high res grid
    shifts=shifts1( frameidx - numframes + 1 : frameidx, : );
    %position of high res grid centered at the average of input frames
    avgxy=mean(shifts);
    shiftswrtavg=(shifts-repmat(avgxy,numframes,1));
    %calculate coordinates of the shifted frame wrt the average high
    %grid, calculate coordinates of overlap region between each frame
    %and the average high res grid
    [corsfrm,corofovr,perovrlp]=boundingbox(shiftswrtavg,hy,hx,LY,LX);
    %sort the percentage overlap to determine the frames that constitute
    least area of overlap
    [so,ind]=sort(perovrlp);
    %initialize the high res grid to zeros
    ztemp=zeros(hy,hx);
    %First check for overlap between the average grid
    %and all the given frames, reconstruct the overlapping region.
    %Start eliminating the frame constituting least area of overlap
    %and fill in the left over region, excluding the region already reconstructed.
    count=0;
    for i=0:nfrms-1
        %index of frames after eliminating the frame
        %contributing least area of overlap
        si=sort(ind(steparray(i+1)+1:numframes));
        %compute coordinates of the region to be reconstructed
        %flag =1 indicates overlap
        [flag,u,d]=ovrlprgn(corofovr(:, :,si));
        newcor(:, :,i+1)=[u;d];%coordinates of the region to be filled
        %check for the number of neighbors
        %required for the nonuniform interpolation
        if(maxNN==minNN)
            NN=minNN;
        else

```

```

if(length(si)>=maxNN)
NN=maxNN;
else
NN=length(si);
end
end
%check for overlap
if(flag==1)
%count = 1 is the first reconstruction
%and is treated as a special case as it does not require step by
step reconstruction
count=count+1;
%frames and the corresponding shifts after eliminating the frame
%constituting the least area of overlap
innew=in(:,si);
shiftsnew=shifts(si,:);
if(count==1)
%first reconstruction
ztemp=regionbasednonuniforminterp(innew,lrsx,lrsy,LX,LY,shiftsnew,NN,...
newcor(:,i+1),avgxy,ztemp);
oldcor=newcor(:,i+1); %of the region filled
else
%region. Update the coordinates of the reconstructed region after
%each sub-reconstruction.
oldcorcopy=oldcor;
diffcor=newcor(:,i+1)-oldcorcopy; %difference between the new coordinates
and old coordinates
diffcor=diffcor';
c=diffcor(:);
indk=find(c==0);%if the difference is 0 no reconstruction
is needed in the corresponding direction
if(length(indk)~=0)
for k=1:length(indk)
%compute the coordinates for each subsection of the
%region to be filled
switch(indk(k))
case 1
ncor(1,1)=oldcorcopy(1,1)+c(indk(k));
ncor(1,2)=oldcorcopy(1,2);
ncor(2,1)=oldcorcopy(1,1)-1;
ncor(2,2)=oldcorcopy(2,2);
oldcorcopy(1,:)=ncor(1,:);
case 2
ncor(1,1)=oldcorcopy(1,1);
ncor(1,2)=oldcorcopy(1,2)+c(indk(k));

```

```

ncor(2,1)=oldcorcopy(2,1);
ncor(2,2)=oldcorcopy(1,2)-1;
oldcorcopy(1,:)=ncor(1,:);
case 3
ncor(1,1)=oldcorcopy(2,1)+1;
ncor(1,2)=oldcorcopy(1,2);
ncor(2,1)=oldcorcopy(2,1)+c(indk(k));
ncor(2,2)=oldcorcopy(2,2);
oldcorcopy(2,:)=ncor(2,:);
case 4
ncor(1,1)=oldcorcopy(1,1);
ncor(1,2)=oldcorcopy(2,2)+1;
ncor(2,1)=oldcorcopy(2,1);
ncor(2,2)=oldcorcopy(2,2)+c(indk(k));
oldcorcopy(2,:)=ncor(2,:);
end
%reconstruct the subsection of the region and keep
%adding to the previous filled region
ztemp=regionbasednonuniforminterp(innew,lrsx,lrsy,LX,LY,shiftsnew,NN,...
ncor,avgxy,ztemp);
end
oldcor=newcor(:,i+1);
else
oldcor=newcor(:,i+1);
end
end
else
end
end
%store the higres frame
out(:,framedx-numframes+1)=ztemp;
%store coordinates
cord(:,frameidx-numframes+1)=oldcor;
t=oldcor(1,1)-1;
b=hy-oldcor(2,1);
l=oldcor(1,2)-1;
r=hx-oldcor(2,2);
%Perform Wiener filtering on the higres image
%softpad the unfilled borders
%check whether filtering has to be performed
if max(size(HW)) > 1
[wsy,wsx]=size(HW);
wbuff=round((wsy-hy)/2);
zwtemp=real(ifft2(fft2(softpad(ztemp(oldcor(1,1):oldcor(2,1),...
oldcor(1,2):oldcor(2,2)),wbuff+t,wbuff+r,wbuff+b,wbuff+l)).*HW));

```

```
outwiener(:, :, frameidx-numframes+1)=zwtemp(wbuff-1:hy+wbuff-2,...  
wbuff-1:hx+wbuff-2);%crop the result to the size of high res  
else  
outwiener(:, :, frameidx-numframes+1)=out(:, :, frameidx-numframes+1);  
end  
end
```


registershift.m

```
function S = registershift( image1, image2, thresh, buff );
Al Schaum Registration Algorithm using Prewitt Operator. Uses Peleg's iterative estimation technique to treat large shift values.
image1 Reference image
image2 Unknown shift image
thresh The threshold for the largest shift to accept, suggested=.1
buff The border buffer to ignore due to border effects and shift effects, suggested=max of expected shift
S=[sx,sy] Output shift vector
sx horizontal shift
sy vertical shift
Author: Dr. Russell C. Hardie
June 1996, modified 8/12/04

%-----%
% Estimate the discrete gradients %
%-----%
xkernel=(1/6)*[1 0 -1
1 0 -1
1 0 -1];
ykernel=(1/6)*[1 1 1
0 0 0
-1 -1 -1];
gx=conv2(image1,xkernel,'same');
gy=conv2(image1,ykernel,'same');
%-----%
% Cut out center because of later shift effects %
%-----%
[fully,fullx]=size(gx); % old size
gx=gx(buff:fully-buff+1,buff:fullx-buff+1);
gy=gy(buff:fully-buff+1,buff:fullx-buff+1);
image1=image1(buff:fully-buff+1,buff:fullx-buff+1);
[ydim,xdim]=size(image1); % new smaller size
%-----%
% Generate the M matrix %
%-----%
cross=sum(sum(gx.*gy));
M=[ sum(sum(gx.^2)), cross,
cross, sum(sum(gy.^2))];
% Initialize loop constants %
count=0; % Number of times through loop
stop=0; % Loop stop flag
im2=image2; % start with the "warped" image being the full second image
S=zeros(2,1); % set initial shift estimates to zero
```

```

X=[1:fullx]; % samp grid of original data in x
Y=[1:fully]'; % samp grid of original data in y
% Begin iterative estimates %
while stop =1
count=count+1;
% calculate the difference image over the interior region
gt=im2(buff:fully-buff+1,buff:fullx-buff+1)-image1;
% Generate the V matrix
V=[sum(sum( gt.*gx ))
sum(sum( gt.*gy ))];
% Find local shift estimate (Schaum)
local=inv(M)*V;
% Update the global estimate
S=S+local;
% See if its time to stop or warp and continue
if count > 10 | ( abs(local(1)) < thresh & abs(local(2)) < thresh)
stop=1;
else % sub-pixelly shift image2 according to latest estimate
XI=[1-S(1):fullx-S(1)];
YI=[1-S(2):fully-S(2)];
im2=interp2(X,Y,image2,XI,YI,'bilinear');
% Note that this repositioning puts NaN outside of original
% grid. This is why I cut out using buff and only operate on
% the interior portion (free of NaN).
end
end
end

```

boundbox.m

function [corshiftframes,corovrlp,perovrlp]=boundbox(s,hy,hx,LX,LX);
Calculates the upper left and bottom right coordinates of the shifted frame with respect to the reference frame. Also computes the upper left and bottom right coordinates of the overlap region between each frame and the reference average grid.
s is the shift between the frames and the average high res grid
LY,LX is the upsampling factor along Y,X
corshiftframes are the coordinates of the shifted frames. size of corshiftframes is 2x2
corovrlp are the coordinates of the overlap region between each shifted frame and the reference frame. size of corovrlp is 2x2
perovrlp is the percentage overlap of each frame with the reference frame
Author: Balaji Narayanan
Date: 10/27/04

```
%size
[sy,sx,nfrms]=size(s);
%initialize coordinates
corshiftframes=zeros(2,2,nfrms); corovrlp=zeros(2,2,nfrms);
ncor(:,1)=1;
%calculate coordinates and percentage overlap
for i=1:sy
    %shifts in terms of high res pixel spacing
    sy1=round(LY*s(i,2)); sx1=round(LX*s(i,1));
    %coordinates
    yu=1+sy1; yd=hy+sy1; xu=1+sx1; xd=hx+sx1;
    corshiftframes(:,i)=[yu,xu;yd,xd];
    ncor(:,2)=corshiftframes(:,i);
    %compute percentage of overlap
    [flag,u,d]=ovrlprgn(ncor); corovrlp(:,i)=[u;d];
    perovrlp(i)=100*prod(abs(diff(corovrlp(:,i),1,1))+1)/(hy*hx);
end
```

ovrlprgn.m

function [flag,ul,br]=ovrlprgn(ulbrcoordinates);

Checks for the overlap between the given frames. Calculates the upper left corner and bottom right corner coordinates of the region of overlap between the frames.

flag =1 indicates there is overlap, flag=0 indicates no overlap region

ulbrcoordinates contains the upper left corner and bottom right corner coordinates of the each frame wrt the average grid. It is a matrix of size(2,2,z) if z==1 the returned coordinates are the coordinates of the overlap region between the given frame and the average grid.

ul is the coordinate of upperleft corner of the region

br is the coordinate of bottom right corner of the region

size of ul and br is 1x2

Author: Balaji Narayanan

Date: 10/27/04

%initialize

ul=0; br=0;

%size

[sy,sx,sz]=size(ulbrcoordinates);

%compute coordinates

upperleftcor=ulbrcoordinates(1,:,1:sz);bottomrightcor=ulbrcoordinates(2,:,1:sz);

ul=max(upperleftcor,[],3); br=min(bottomrightcor,[],3);

%check for overlap

if((br(1,1)>=ul(1,1))&(br(1,2)>=ul(1,2)))

flag=1;

else

flag=0;

end

regionbasednonuniforminterp.m

```
function z=regionbasednonuniforminterp(yobs,lrsx,lrsy,LX,LY,S,...
NN,ulbrcord,avgxy,z)
```

Region based translational shift weighted neighbor(linear)interpolator for multiple frames. Takes multiple frames (yobs) and estimates the proper values on a specific region of high resolution grid (z) defined by the coordinates in "ulbrcord". The high res grid is centered about the ****AVERAGE**** x,y position of the input frames.

z High resolution estimate of specific portion on uniform grid

yobs low resolution images yobs is matrix of size lrsy X lrsx X nframes]

lrsx Horizontal low res image size

lrsy Vertical low res image size

LX The desired upsample factor in X

LY The desired upsample factor in Y

S translational shifts for each frame measures in low

resolution pixel spacings S=[sx1,sy1; sx2,sy2; ...]

ulbrcord The upperleft corner coordinates and the bottom right right corner coordinates of the region to be reconstructed

NN Number of neighbors to weight for each point (1 - number of frames). 1=nearest neighbor.

Author 1: Dr. Russell Hardie 9/17/96

Author 2: modified by Balaji Narayanan 10/30/04

University of Dayton, All Rights Reserved

```
% size of high res image
```

```
hy=lrsy*LY; hx=lrsx*LX;
```

```
%upper left and bottom right corner coordinates
```

```
uy=ulbrcord(1,1); dy=ulbrcord(2,1); ux=ulbrcord(1,2);
```

```
dx=ulbrcord(2,2);
```

```
%number of pixels along and Y and X direction
```

```
%to be filled
```

```
npv=dy-uy+1; npu=dx-ux+1;
```

```
%vy and vx is the loop size along Y and X, if the number of pixels are greater than the upsample factor, clip the loop size to LY and LX
```

```
if(npv>=LY)
```

```
npv=LY;
```

```
else
```

```
npv=npv;
```

```
end if(npu>=LX)
```

```
npu=LX;
```

```
else
```

```
npu=npu;
```

```
end
```

```
%determine the position of the upper y and x coordinate relative to the upper left corner of the high res grid within 1 to LY and 1 to LX
```

```
psy=mod(uy,LY); psx=mod(ux,LX);
```

```

%clip the relative position to the upsample factor, if it is greater than the upsample factor
if(psy==0)
psy=LY;
else
psy=psy;
end if(psx==0)
psx=LX;
else
psx=psx;
end
%store initial relative position along x
psxcopy=psx;
% determine how many low res frames and number of low res pixels
[ly,lx,nframes]=size(yobs);
% initialize index storage matrix
index=zeros(nframes,2);
%-----%
% fill in the overlap region based on the starting %
% and the terminating coordinates of the region %
%-----%
for j=1:npy
%pixel position along y to be filled in the high res grid
ny=[uy+j-1:LY:dy];
%starting pixel and ending pixel
starty=ceil(ny(1)/LY); endy=ceil(ny(length(ny))/LY);
for i=1:npx
%pixel position along x to be filled in the high res grid
nx=[ux+i-1:LX:dx];
% high resolution pixel position with respect to avgxy
pos=[(psx-1)/LX,(psy-1)/LY]+avgxy;
%-----%
% Find the closest sample from each frame to pos %
%-----%
for k=1:nframes
% find the position of pos with respect to grid of frame k
posk=pos-S(k,:);
% round this off to find the index associated with the
% nearest sample from grid k
index(k,:)=round(posk);
% find the error distance for this nearest sample
% (i.e., how much rounding off is required).
d(k)=sqrt(sum([posk-index(k,:)].^2));
end
% Sort the distances for each frame
% (i.e., the amount rounding off required for each frame)

```

```

[sdist,si]=sort(d);
% Loop over the closest NN points and
% Generate the weighted sum portion of the frame for insertion.
%starting and ending pixels along X
startx=ceil(nx(1)/LX);
endx=ceil(nx(length(nx))/LX);
% initialize the weighted frame
frame=zeros(endy-starty+1,endx-startx+1);
% reciprocal distance vector
sdist=clip(sdist,.001,inf); sdistr=1./sdist;
%sum of the reciprocal distance vector of the NN neighbors
tot1=sum(sdistr(1:NN));
for m=1:NN
% isolate frame number si(m)
temp=yobs(:,si(m));
% do integer shift of image as required by
temp=circshift3(temp,index(si(m),1),index(si(m),2));
%select the right portion of the low res frame
temp1=temp(starty:endy,startx:endx);
% add to running sum
frame=frame+sdistr(m)*temp1;
end
% insert weighted frame into right portion of the high-res grid
z(ny,nx)=(frame(1:1:length(ny),1:1:length(nx)))/tot1;
%increment the relative position along X
%or clip to LX
if(psx==LX)
psx=1;
else
psx=psx+1;
end
end
%reset the relative position to
%the initial value
psx=psxcopy;
%increment the relative position along Y
%or clip it to LY
if(psy==LY)
psy=1;
else
psy=psy+1;
end
end
end

```

clip.m

```
function C=clip(c,minn,maxx)
Clipping function. Takes any c (scalar, vector or matrix) and clips each value with the
limits of minn and maxx.
C Clipped version of c
c Input scalar, vector or matrix.
minn Minimum value of C
maxx Maximum value of C
Author: Dr. Russell C. Hardie
Date: 9/23/96
Copyright University of Dayton, All rights reserved

% initialize output to be input
C=c;
% find all positions with values below minn
Imin=find(c<minn);
% find all positions with values above maxx
Imax=find(c>maxx);
% Replace all sub minn values with minn
C(Imin)=minn*ones(size(Imin));
% Replace all above maxx values with maxx
C(Imax)=maxx*ones(size(Imax));
```


circhshiftm.m

```
function [frame] = circhshift3(frame,shx,shy);
Shifts an image by shx,shy. The new image data entering the field of view (FOV) is the
repeated border data.
frame The shifted image and image to be shifted
shx horizontal shift in terms of # of pixels
shy vertical shift in terms of # of pixels

[ydim,xdim]=size(frame);
% Shift the image by sx,sy
% Horizontal shifts
if shx > 0
% if positive, repeat right border and attach
rs=frame(:,xdim);
frame=[frame(:,shx+1:xdim),rs*ones(1,shx)];
elseif shx < 0
% if negative, mirror left side strip and attach
ls=frame(:,1);
frame=[ls*ones(1,abs(shx)),frame(:,1:xdim-abs(shx))];
end
% Vertical shifts
if shy > 0
% if positive, mirror bottom strip and attach
bt=frame(ydim,:);
frame=[frame(shy+1:ydim,:) ; ones(shy,1)*bt];
elseif shy < 0
% if negative, mirror top strip and attach to top.
tp=frame(1,:);
frame=[ones(abs(shy),1)*tp ; frame(1:ydim-abs(shy),:)];
end
```

nonuniforminterpolation.m

function out = nonuniforminterpolation(in, LX, LY, shifts, numneighbors)
Translational shift weighted neighbor (linear) interpolator for multiple frames forming a nonuniform grid. Takes multiple frames (in) and estimates the proper values on the high resolution grid (out). The high res grid is centered about the *AVERAGE* x,y position of the input frames.

out High resolution estimate on uniform grid

in Input frames

LX The desired upsample factor in X

LY The desired upsample factor in Y

shifts translational shifts for each frame measures in low resolution pixel spacings shifts = [sx1,sy1; sx2,sy2; ...];

numneighbors Number of neighbors to weight for each point
(1 - number of frames). 1 = nearest neighbor.

Author: Dr. Russell C. Hardie

9/17/96, modified 8/12/04

```
% Get all size info %  
[ lrsy, lrsx, numframes ] = size(in );  
% size of high res image  
sy = lrsy * LY; sx = lrsx * LX;  
% form hires grid and fill with zeros  
out = zeros( sy, sx );  
% initialize index storage matrix  
index=zeros( numframes, 2 );  
% find average x,y position  
avgxy = mean( shifts );  
% Loop over a representative portion of the image %  
for j=1:LY  
for i=1:LX  
% high resolution pixel position with respect to avgxy  
pos=[(i-1)/LX,(j-1)/LY]+avgxy;  
% Find the closest sample from each frame to pos %  
for k = 1 : numframes  
% find the position of pos with respect to grid of frame k  
posk = pos - shifts( k, : );  
% round this off to find the index associated with the  
% nearest sample from grid k  
index( k, : ) = round( posk );  
% find the error distance for this nearest sample  
% (i.e., how much rounding off is required).  
d(k)=sqrt(sum([posk-index(k,:)].^2));  
end
```

```

    % Sort the distances for each frame
    % (i.e., the amount rounding off required for each frame)
    [sdist,si]=sort(d); % Insert the weighted values into the high resolution grid %
    % Loop over the closest NN points and
    % Generate the weighted sum frame for insertion.
    % Initialize the weighted frame
    frame = zeros( lrsy, lrsx );
    % Reciprocal distance vector
    sdist = clip( sdist, .001, inf ); sdistr = 1 ./ sdist;
    tot = sum( sdistr( 1 : numneighbors ) );
    for m = 1 : numneighbors
        % Do integer shift of image as required by
        % index(si(m))
        temp = circshift3( in(:, :, si(m)), index(si(m),1), index(si(m), 2 ) );
        % Add to running sum
        frame = frame + sdistr( m ) * temp;
    end
    % Insert weighted frame into high-res grid
    out( j : LY : sy, i : LX : sx ) = frame / tot;
end
end

```

wienerfilter.m

```
function wienerout=wienerfilter(in,pad,nsr,uy,ux);  
Filters the input using inverse wiener filter response  
in input image  
pad buffer to avoid border effects  
uy ux upsampling factor along y and x  
%nsr noise-to-signal ratio  
  
[hy,hx]=size(in);  
% default system PSF  
psf = ones( uy, ux) / ( uy * ux );  
% DFT of PSF  
H = fft2( psf, hy + 2 * pad, hx + 2 * pad);  
% DFT of Wiener filter  
absH2 = ( abs( H ).^2); HW = conj( H )./ ( absH2 + nsr );  
[wsy1,wsx1]=size(HW); wbuff=round((wsy1-hy)/2);  
%wiener filtering  
temp=real(ifft2(fft2(softpad(in,wbuff,wbuff,wbuff,wbuff)).*HW));  
wienerout=temp(wbuff-1:hy+wbuff-2, wbuff-1:hx+wbuff-2);
```

APPENDIX C

MATLAB SOURCE CODE FOR PWS-BASED SR OF VIDEO

numfilters.m

Code to plot the number of filters required for $M=10$ partitions as a function of the observation window size.

```
N=[2:1:15]; M=10; n2=N.^2; num=M*(2.^(n2));  
h=figure;  
semilogy(N,num,'o-'); set(get(h,'CurrentAxes'),'FontSize',10);  
xlabel('N','FontSize',18); ylabel('Number of filters','FontSize',18);  
grid on; %print numfilt.eps
```

flopcount.m

Code to plot the flop count per output pixel as a function of HR grid density fraction. The flop count is plotted for translational motion and for arbitrary motion.

```
%clear
close all; clear all; clc;
%initialize variables
s1=15; s2=15; p=linspace(0.1,1,10); l1=5;
l2=5; k1=5; k2=5; m1=64; m2=64; nk1=7; nk2=7; nl1=7; nl2=7;
%compute flops according to Eq. 7.3 and 7.4 in
%dissertation with M=1 and discarding first term
n3=((p*s1*s2).^3)/3; n2=2*k1*k2*((p*s1*s2).^2); n=p*s1*s2;
numpixels=ceil((m1*l1-s1+1)/k1)*ceil((m2*l2-s2+1)/k2)*k1*k2;
numpixels1=ceil((m1*nl1-s1+1)/k1)*ceil((m2*nl2-s2+1)/k2)*k1*k2;
tem=((n3+n2)/numpixels);
%for the case l1=k1,l2=k2
nflops=tem+n;
%fixing l1 and l2 and changing k1 and k2
tem1=(n3+2*nk1*nk2*((p*s1*s2).^2))/numpixels; nflops1=l1*l2*tem1+n;
%fixing k1 and k2 and changing l1 and l2
tem2=((n3+n2)/numpixels1); nflops2=nl1*nl2*tem2+n;
%for rotation
rn3=n3./(k1*k2); rn2=2*((p*s1*s2).^2); rotflops=rn3+rn2+n;
%figure for translation
h1=figure; plot(p,nflops,'o-',p,repmat(229,size(p)),'*-',p,nflops2,'-v');
set(get(h1,'CurrentAxes'),'FontSize',12);
xlabel('HR grid density fraction f','FontSize',18);
ylabel('Flops per output pixel','FontSize',18);
legend('PWS SR filters  $K_1 = K_2 = 5$  and  $L_1 = L_2 = 5$ ','WNN',...
'PWS SR filters  $K_1 = K_2 = 5$  and  $L_1 = L_2 = 7$ ',2);
axis tight; grid on
%print flops2.eps
%rotation figure
h2=figure; semilogy(n,rotflops,'-v',n,nflops,'-o');
set(get(h2,'CurrentAxes'),'FontSize',12);
xlabel('HR grid density fraction f','FontSize',18);
ylabel('Flops per output pixel','FontSize',18);
legend('Arbitrary Motion','Translation',2); axis tight; grid on;
%print flopsrot.eps;
```

runsimsuperres.m

Code to generate the HR images using WNN, Delaunay and partition based interpolation techniques. This code takes the simulated LR frames as input and plots the error for the various techniques.

```
close all; clear all; clc;
%load visible image and autocorrelation
%and cross correlation matrices
load kenaerial2.mat load RP5block15n10.mat; load code7by7.mat;
load partition7by710RP5block15n10.mat;
%size of visible image for generating low res frames
ny=324;nx=324;
%upsampling factor (blurring factor for
%generating low res frames)
uy=5; ux=5;
%number of neighbors
nn=4;
%variance of noise
noisevar=10;
%visible image
data=kenaerial2(1:ny,189:189+nx-1);
%generate noisy and blurred low resolution frames and true high res image
[simlowobs,truedata]=simulatedata(data,ny,nx,uy,ux,noisevar);
[lrsy,lrsx,numframes]=size(simlowobs);
%high res grid size
hix = ux*lrsx; hiy = uy*lrsy;
>window size
wsy=15; wsx=15; dwx=5; dwy=5; pwy=7; pwx=7;
%gamma noise to signal ratio
nsr=linspace(0.01,0.15,15);
%cell array to store indices of frames
index=cell(uy*ux,1);
%select the frames
index = [1];[1 13];[1 13 25];[1 3 13 25];[1 11 13 15 25];[1 9 11 13 15 25];[1 9 11 13 15 17 25];[1 7 9 11 13 15 17 25]; [1 7 9 11 13 15 17 19 25];[1 5 7 9 11 13 15 17 19 25];[1 5 7 9 11 13 15 17 19 21 25];[1 3 5 7 9 11 13 15 17 19 21 25]; [1 3 5 7 9 11 13 15 17 19 21 23 25];[1 2 3 5 7 9 11 13 15 17 19 21 23 25];[1 2 3 5 7 9 11 13 15 17 19 21 23 24 25]; [1 2 3 4 5 7 9 11 13 15 17 19 21 23 24 25];[1 2 3 4 5 7 9 11 13 15 17 19 21 22 23 24 25];[1 2 3 4 5 6 7 9 11 13 15 17 19 21 22 23 24 25];[1 2 3 4 5 6 7 8 9 11 13 15 17 18 19 20 21 22 23 24 25];[1 2 3 4 5 6 7 8 9 10 11 13 15 16 17 18 19 20 21 22 23 24 25];[1 2 3 4 5 6 7 8 9 10 11 12 13 15 16 17 18 19 20 21 22 23 24 25];[1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25]; %shifts for the frames
shfts=[0,0;0,0.2;0,0.4;0,0.6;0,0.8;0.2,0;0.2,0.2;0.2,0.4;0.2,0.6;0.2,0.8;0.4,0;0.4,0.2;...
0.4,0.4;0.4,0.6;0.4,0.8;0.6,0;0.6,0.2;0.6,0.4;0.6,0.6;0.6,0.8;0.8,0;0.8,0.2;0.8,0.4;
```

```
0.8,0.6;0.8,0.8];
```

```

    avg=[0 0]; %position of reference frame
    mpos=[5 5]; %starting position on the overlapping portion of the grid
    %portion of the true image being estimated
    start=mpos;
    true=truedata(start(1,1):uy*lrsy,start(1,2):ux*lrsl);
    [sz1,sz2] =size(true);
    for j=1:uy*ux
        %select image sequence
        lowobs = simlowobs(:, :,indexj,1);
        %size info for selected sequence
        [lrsy,lrsx,numframes]=size(lowobs);
        %number of neighbors
        if(numframes<=nn)
            numneighbor=numframes;
        else
            numneighbor=nn;
        end
        %shifts corresponding to selected frames
        shifts=shfts(indexj,1,:);
        %weighted neighbor nonuniform interpolation
        z=zeros(hiy,hix);
        z=regionbasednonuniforminterp(lowobs,lrsy,lrsx,uy,ux,...
            shifts,numneighbor,[mpos(1,1)
            mpos(1,2);hiy hix],avg,z); wnn=z(mpos(1,1):hiy,mpos(1,2):hix);
        %overlapping region on the high res grid
        [hrgrid,ovrpos]=generatehrgridtrans(lowobs,shifts,uy,ux,0,0);
        ovrpgrid=hrgrid(mpos(1,1):end,mpos(1,2):end);
        %partition interpolation with M=1
        wienerinterp=blockprocess(ovrpgrid,wsy,wsx,dwy,dwx,R5by5,P5by5,1);
        %partition interpolation output
        %precompute weights
        W=wgthsforpartition(ovrpgrid,wsy,wsx,R5by5p,P5by5p);
        %interpolate using partition filters
        partwienerinterp = partitionblockprocessforovrlp(ovrpgrid,wsy,wsx,pwy,pwx,...
            dwy,dwx,code7by7,W);
        %deluanay interpolation
        zi=triangtrans(lowobs,uy,ux,shifts,[mpos(1,1) mpos(1,2);hiy hix]);
        %indices
        indy2=[1:dwy:sz1-wsy+1]; indy2=[1:dwy:sz1-wsy+1];
        %top left coordinates
        topleftcord = [1,1;1+floor((wsy-dwy)/2),1+floor((wsx-dwx)/2)];
        %bottom right coordinates
        botrightcord = [sz1,sz2;indy2(length(indy2))+dwy-1+ceil((wsy-dwy)/2),...

```



```

indx2(length(indx2))+dwx-1+ceil((wsx-dwx)/2));
% max and min
[zy,zx]=size(topleftcord);
maxtopleftcor = max(topleftcord,[],1);
minbotrightcor = min(botrightcord,[],1);
%indices for cropping the results to the same size
stlindex = repmat(maxtopleftcor,zy,1)-topleftcord + repmat([1,1],zy,1);
sbtindex = botrightcord-repmat(minbotrightcor,zy,1);
%crop the true image and results
true1 = true(stlindex(1,1):end-sbtindex(1,1),stlindex(1,2):end-sbtindex(1,2));
wnn1 = wnn(stlindex(1,1):end-sbtindex(1,1),stlindex(1,2):end-sbtindex(1,2));
zi1 = zi(stlindex(1,1):end-sbtindex(1,1),stlindex(1,2):end-sbtindex(1,2));
hres1 = wienerinterp(stlindex(2,1):end-sbtindex(2,1),...
stlindex(2,2):end-sbtindex(2,2));
hres2 = partwienerinterp(stlindex(2,1):end-sbtindex(2,1),...
stlindex(2,2):end-sbtindex(2,2));
%find gamma such that error is minimum
for count=1:length(nsr)
wienerwnn=wienerfilter(wnn1,20,nsr(count),uy,ux);
[ae(count),mse(count),snr(count),avgerr(count)] = dif(true1,wienerwnn);
wienerdnt=wienerfilter(zi1,20,nsr(count),uy,ux);
[aet(count),mset(count),snrt(count),avgerrt(count)] = dif(true1,wienerdnt);
end
%store the errors
maewnn(j)=min(ae); msewnn(j)=min(mse);
maednt(j)=min(aet); msednt(j)=min(mset);
[maepf1(j),msepf1(j),snrpf1,avgerrpf1]=dif(true1,hres1);
[maepf10(j),msepf10(j),snrpf10,avgerrpf10]=dif(true1,hres2);
%store the high res images
hreswnn(:,j)=wienerfilter(wnn1,20,nsr(find(ae==min(ae))),uy,ux);
hresdnt(:,j)=wienerfilter(zi1,20,nsr(find(aet==min(aet))),uy,ux);
hrespf1(:,j)=hres1; hrespf10(:,j)=hres2;
end
ind=[1:1:25];
h1=figure;
plot(ind,maewnn,'r-x',ind,maednt,'k-',ind,maepf1,'b*',ind,maepf10,'c-');
set(get(h1,'CurrentAxes'),'FontSize',10);
xlabel('Number of frames available','FontSize',18);
ylabel('Mean absolute error','FontSize',18);
legend('WNN','Delaunay','PWS SR filters M=1','PWS SR filters M=10',1);
% print maerror3.eps
%Display images
%original true image
imstdf(data);
%LR frame

```

```

imstdf(lowobs(:,:,1));
%partially populated HR grid
imstdf(hrgrid)
%processed outputs
imstdf(hreswnn(:,:,5)); imstdf(hresdnt(:,:,5));
imstdf(hrespfl(:,:,5)); imstdf(hrespfl0(:,:,5));
istdf(imresize(lowobs(:,:,1),LX,'bicubic') );

```

simulatedata.m

```
function [simlowobs, truedata]=simulatedata(data,ny,nx,uy,ux,noisevar);  
Generates blurred and nosiy low resolution frames from single image  
simlowobs nosiy and blurred low resolution frames  
truedatatrue low res data data high resolution image  
ny nx size of data  
uy ux detector size to blur  
noisevar variance of noise  
Author : Balaji Narayanan  
date 03/27/05
```

```
in =data;  
%noisepower  
noisepow = sqrt(noisevar);  
buy1=floor((uy-1)/2); buy2=ceil((uy-1)/2);  
bux1 = floor((ux-1)/2); bux2 = ceil((ux-1)/2);  
%psf  
psf=ones(uy,ux)./(uy*ux);  
%crop true test data same size as the blurred image (256,256)  
truedata = in(buy1+1:ny-buy2,bux1+1:nx-bux2);  
obs1=conv2(in,psf,'valid');  
obs=conv2(in,psf,'valid') + randn(size(truedata)).*noisepow;  
[sy,sx]=size(obs); block = im2col(obs,[uy,ux],'distinct');  
for i=1:uy*ux  
simlowobs(:,i) = reshape(block(i,:),sy/uy,sx/ux); end
```

generatehrgridtrans.m

```
function [hrgrid,ovrlpos]=generatehrgridtrans(lowobs,shifts,...
uy,ux,pos,flag);
Generates the HR grid by inserting the samples from LR frames using the shifts
hrgrid high resolution grid
shifts motion parameters
lowobs LR frames
pos position where the HR grid is centered pos=0 HR grid is centered at 0,0 if pos=1 hr
grid is centered about the average of the shifts
flag flag=1 overlapping region of the HR grid flag=0 HR grid with
high resolution size

%size info
[ly,lx,numframes]=size(lowobs); hiy=ly*uy; hix=lx*ux;
%low res shifts for the current block of frames
shiftslow =shifts;
frame = lowobs; %current block of frames
%shifts interms high res pixel spacing
shiftshigh=uy*shiftslow;
if(pos==0)
%position of hig res grid is centered at 0,0
meanhighshifts=[0,0];
else
%position of high res grid centered at average
%position of shifts
meanhighshifts=mean(shiftshigh);
end
%position of low res frames with respect to the position of the
%high res grid
newshifts=shiftshigh-repmat(meanhighshifts,numframes,1);
%round off to find the index associated with the high res grid
rndshifts=round(newshifts);
%minimum of the rounded shifts
minrndshifts=min(rndshifts,[],1);
%fix the minimum as [1,1] and
%all other pixels relative to [1,1]
pixelpos=rndshifts+repmat([1 1]-minrndshifts,numframes,1);
% maximum pos is the starting coordinate of the overlap region
maxpos=max(pixelpos,[],1)
%starting and ending coordinates to select the HR grid
if(maxpos(1,1)~=ux)
endx=ux*lx; sx1=1;
else
endx=maxpos(1,1)+hix-1; sx1=1-minrndshifts(1,1);
end
```

```

if(maxpos(1,2)~=uy)
endy=uy*ly; sy1=1;
else
endy=maxpos(1,2)+hiy-1;sy1=1-minrndshifts(1,2);
end
%insert each lowres frame into the full high res grid
%frames inserted into the same position are averaged
framecount=zeros(endy,endx); grid = zeros(endy,endx);
szg=size(grid);
for i=1:numframes
%high res pixel positions
col=[pixelpos(i,1):ux:(lx-1)*ux+pixelpos(i,1)];
row=[pixelpos(i,2):uy:(ly-1)*uy+pixelpos(i,2)];
grid(row,col)=grid(row,col)+frame(1:length(row),1:length(col),i);
framecount(row,col)=framecount(row,col)+1; end
framecount(find(framecount(:)==0))=1; hgrid=grid./framecount;
%replace all 0 with -1
indexn=find(hgrid(:)==0);hgrid(indexn)=-1;
if(flag==1)
%overlapping region of the grid
hrgrid=hgrid(maxpos(1,2):hiy,maxpos(1,1):hix);
else
%full high res grid
hrgrid=hgrid(sy1:hiy+sy1-1,sx1:hix+sx1-1);
end
%coordinates
ovrlpos=[maxpos(1,2)-sy1+1,maxpos(1,1)-sx1+1;hiy-sy1+1,hix-sx1+1];

```

blockprocess.m

```
function out=blockprocess(in,wsy,wsx,dwy,dwx,Rmatrix,Pmatrix,ovrlpflag);
Takes a high resolution grid and performs block processing to interpolate and fill the grid.
Uses one partition.
in High resolution grid
wsy, wsx Observation window size
dwy, dwx Desired window size within the observation window to be filled
(dwxi=wsx) (dwyi=wsy)
Rmatrix Correlation matrix
Pmatrix Expected value of the product of desired and observed
ovrlpflag 1 fills overlapping region of the grid efficiently without
for loops with dwy=uy and dwx=ux
0 uses for loops for filling the grid
out processed output
Author : Balaji Narayanan
Date : 02/08/05

%size info of the high res grid
[sy,sx]=size(in);
% when desired window size is same
% as observed window size
if((dwy==wsy)&(dwx==wsx))
%remainder
remy = mod(sy,wsy);
remx = mod(sx,wsx);
%y and x index to crop the input
%such that the size is a multiple of wsy,wsx
suby1 = 1 + floor(remy/2);
suby2 = sy-ceil(remy/2);
subx1 = 1+floor(remx/2);
subx2 = sx-ceil(remx/2);
%crop the input
subin = in(suby1:suby2,subx1:subx2);
% create distinct moving blocks
block = im2col(subin,[wsy,wsx],'distinct');
%output size
[szy,szx] = size(subin);
else
%create an array of wsy X wsx moving data
blocktmp = im2col(in,[wsy,wsx],'sliding');
%create indices to select appropriate
%blocks seperated by dwy x dwx
ind = 0;
indx = [1:dwx:sx-wsx+1];
indy = [1:dwy:sy-wsy+1]; lenx = length(indx); leny = length(indy);
```

```

for i = 1:lenx
ind = [ind,1+(sy-wsy+1)*(indx(i)-1):dwy:(sy-wsy+1)+...
(sy-wsy+1)*(indx(i)-1)];
end
ind = ind(2:end);
%obtain the blocks
block = blocktmp(:,ind); %output size
szy = dwy*leny; szx = dwx*lenx;
end
if(ovrlpflag =1)
%size of block
[by,bx]=size(block);
%allocate memory to store block processed data
temp = zeros(dwy*dwx,bx); tic
%process block by block
for count=1:bx
%find index of non zero elements
index=find(block(:,count) ==-1);
%select the proper correlation matrix
R=Rmatrix(index,index);
%select the proper desired matrix
P=Pmatrix(index,:);
%obtain weights for interpolation
w1 = R\ P; [ws1,ws2]=size(w1); w2=sum(w1,1); %normalize wghts(wghts are close to 1)
w=w1./(repmat(w2,ws1,1));
%process and the store the result
temp(:,count)=((w')*block(index,count));
end
tblk=toc
else
tic
%find index of non zero elements index=find(block(:,1) ==-1);
%select the propoer correlation matrix
R=Rmatrix(index,index);
%select the proper desired matrix
P=Pmatrix(index,:);
%obtain weights
w1 = R\ P; [ws1,ws2]=size(w1); w=w1./(repmat(w2,ws1,1));
%process and the store the result
temp = ((w')*block(index,:));
tblktoc=
end
%form the output
out=col2im(temp,[dwy,dwx],[szy,szx],'distinct');

```

wghtsforpartition.m

```
function W=wghtsforpartition(hrgrid,wsy,wsx,R,P);
Computes the weight matrix from the autocorrelation and cross correlation matrix for all
the partitions based on a specific configuration of the pixels within the observation window.
hrgrid overlapping portion of the highres grid
wsy wsx observation window size
R P autocorrelation and cross correlation matrix
W weight matrix for all the partitions
Author: Balaji Narayanan
Date: 04/26/05

%observation window of size wsy X wsx
temp=hrgrid(1:wsy,1:wsx);
%missing entries indx=find(temp(:) ==-1);
%size info of autocorrelation matrix
[sy,sx,sz]=size(R);
%compute weights for all the partitions
for i=1:sz
W(:,i)=R(indx,indx,i)\P(indx,:,i);
end
```


triangtrans.m

```
function [zi]=triangtrans(yobs,LX,LY,R,gridpoints)
Fills a uniform grid using griddata.m which is based on
Delaunay triangulation of the data.
zi High resolution estimate on uniform grid filled
yobs Low resolution images
lrsx Horizontal low res image size
lrsy Vertical low res image size
LX The desired upsample factor in X
LY The desired upsample factor in Y
R Registration info for each frame S=[sx1,sy1,rot1; sx2,sy2,rot2; ...];
First frame must be 0,0,0.(shifts in low-res pixel spacings,
rotation in degrees about center).
gridpoints points on the grid to be interpolated
Author: Dr. Russell C. Hardie 6/4/97
modified by Balaji Narayanan

% Get all size info %
% determine how many low res frames and number of low res pixels
[lrsy,lrsx,nframes]=size(yobs);
% size of high res image
sy=lrsy*LY; sx=lrsx*LX;
% convert shifts to high-res pixel spacings
R(:,1)=R(:,1)*LX;R(:,2)=R(:,2)*LY;
% specify reference grid points on high res grid
xr=[1:LX:sx];yr=[1:LY:sy];[Xr,Yr] = meshgrid(xr,yr);
% specify the desired grid
xd=[gridpoints(1,2):gridpoints(2,2)];
yd=[gridpoints(1,1):gridpoints(2,1)]; [Xd,Yd] = meshgrid(xd,yd);
% fill the new frame grid X,Y to proper size, will overwrite later
X=Xr; Y=Yr;
zall=[]; Xall=[]; Yall=[];
% Round off and insert samples from each frame %
for i=1:nframes
% Compute all the sample points locations for this frame
X(:)= Xr(:)+R(i,1); Y(:)= Yr(:)+R(i,2);
% add to previous frame info
Xall=[Xall;X(:)]; Yall=[Yall;Y(:)];
frame=yobs(:,i); zall=[zall;frame(:)];
end
zi=griddata(Xall,Yall,zall,Xd,Yd);
```

partitionblockprocessforovrlp.m

```
function out=partitionblockprocessforovrlp(input,wsy,wsx,pwy,...
pwx,dwy,dwx,code,W);
Takes a overlapping portion of high resolution grid and performs block processing using
partition filters to interpolate and fill the grid.
Uses the precomputed weight matrix.
input High resolution grid
wsy, wsx Observation window size
pwy pwx Partition window size
code code book for the partitions
dwy, dwx Desired window size within the observation window to be filled
(dwxi=wsx) (dwyi=wsy) W weights for M partitions and for specific configuration of HR
grid
out processed output
Author : Balaji Narayanan
Date : 04/15/05

% when desired window size is same
% as observed window size
if((dwy==wsy)&(dwx==wsx))
[sy1,sx1]=size(input);
%remainder
remy = mod(sy1,wsy);
remx = mod(sx1,wsx);
%y and x index to crop the input
%such that the size is a multiple of wsy,wsx
suby1 = 1 + floor(remy/2); suby2 = sy1-ceil(remy/2);
subx1 = 1+floor(remx/2); subx2 = sx1-ceil(remx/2);
%crop the input
in = input(suby1:suby2,subx1:subx2);
%output size
[szy,szx] = size(in);
else
in=input;
end
%size
[sy,sx] = size(in);
%size of code
[coy,cox] = size(code);
py1 = floor((wsy-pwy)/2); px1 = floor((wsx-pwx)/2);
%create an array of wsy X wsx moving window
blocktmp = im2col(in,[wsy,wsx],'sliding');
%create indices to select appropriate
%blocks seperated by dwy x dwx
ind = 0; indx = [1:dwx:sx-wsx+1]; indy = [1:dwy:sy-wsy+1];
```

```

lenx = length(indx); leny = length(indy);
for i = 1:lenx
ind = [ind,1+(sy-wsy+1)*(indx(i)-1):dwy:(sy-wsy+1)+(sy-wsy+1)*(indx(i)-1)];
end
ind = ind(2:end);
%obtain the blocks
block = blocktmp(:,ind); [bly,blx] = size(block);
%output size
szy = dwy*leny; szx = dwx*lenx;
%portion of the grid containing the partition data
partgrid=in(py1+1:sy-py1,1+px1:sx-px1); [py,px]=size(partgrid);
%create an array of wsy X wsx moving data
parttmp = im2col(partgrid,[pwy,pwx],'sliding');
%create indices to select appropriate
%blocks seperated by dwy x dwx
ind1 = 0; indx1 = [1:dwx:px-pwx+1]; indy1 = [1:dwy:py-pwy+1];
lenx1 = length(indx1); leny1 = length(indy1);
for i = 1:lenx1
ind1 = [ind1,1+(py-pwy+1)*(indx1(i)-1):dwy:(py-pwy+1)+...
(py-pwy+1)*(indx1(i)-1)];
end
ind1 = ind1(2:end);
%obtain the blocks
partblock = parttmp(:,ind1); [pby,pbx] = size(partblock);
%create memory for output
tempres = zeros(dwy*dwx,blx);tic
for count = 1:blx

index = find(partblock(:,count) ==-1);
temp = partblock(index,count); temp = temp-mean(temp);
%normalize replicatetemp = temp*ones(1,cox);
distance = sum((replicatetemp-code(index,:)).^2,1);
[minvalue,minid] = min(distance);
%find index of missing elements
%obtain weights
w1=W(:,minid); [ws1,ws2]=size(w1);
%weights are close to unity
%normalize the weights such that they sum to 1
w2=sum(w1,1); w3=(1-w2)./(ws1); w=w1+repmat(w3,ws1,1);
%process and the store the result
tempres(:,count) = ((w')*block(index1,count));
end
tblkp=toc
%form the output

```

```
out=col2im(tempres,[dwy,dwx],[szy,szx],'distinct');
```

partitionblockprocess.m

```
function out=partitionblockprocess(input,wsy,wsx,pwy,pwx,dwy,...
dwx,code,Rmatrix,Pmatrix);
Takes a high resolution grid and performs block processing using partition filters to inter-
polate and fill the grid. Used for rotation. uses for loop to implement the filters.
input High resolution grid
wsy, wsx Observation window size
pwy pwx Partition window size
code code book for the partitions
dwy, dwx Desired window size within the observation window to be filled
(dwxi=wsx) (dwyi=wsy)
Rmatrix Correlation matrix
Pmatrix Expected value of the product of desired and observed
out processed output
Author : Balaji Narayanan
Date : 02/15/05

% when desired window size is same as observed window size
if((dwy==wsy)&(dwx==wsx))
[sy1,sx1]=size(input);
%remainder
remy = mod(sy1,wsy); remx = mod(sx1,wsx);
%y and x index to crop the input
%such that the size is a multiple of wsy,wsx
suby1 = 1 + floor(remy/2); suby2 = sy1-ceil(remy/2);
subx1 = 1+floor(remx/2); subx2 = sx1-ceil(remx/2);
%crop the input
in = input(suby1:suby2,subx1:subx2);
%output size
[szy,szx] = size(in);
else
in=input;
end
%size
[sy,sx] = size(in);
%size of code
[coy,cox] = size(code);
py1 = floor((wsy-pwy)/2); px1 = floor((wsx-pwx)/2);
%create an array of wsy X wsx moving window
blocktmp = im2col(in,[wsy,wsx],'sliding');
%create indices to select appropriate
%blocks seperated by dwy x dwx
ind = 0; indx = [1:dwx:sx-wsx+1]; indy = [1:dwy:sy-wsy+1];
lenx = length(indx); leny = length(indy);
for i = 1:lenx
```

```

ind = [ind,1+(sy-wsy+1)*(indx(i)-1):dwy:(sy-wsy+1)+...
(sy-wsy+1)*(indx(i)-1)];
end
ind = ind(2:end);
%obtain the blocks
block = blocktmp(:,ind); [bly,blx] = size(block);
%output size
szy = dwy*leny; szx = dwx*lenx;
%portion of the grid containing the partition data
partgrid=in(py1+1:sy-py1,1+px1:sx-px1); [py,px]=size(partgrid);
%create an array of wsy X wsx of the partition data
parttmp = im2col(partgrid,[pwy,pwx],'sliding');
%create indices to select appropriate blocks separated by dwy x dwx
ind1 = 0; indx1 = [1:dwx:px-pwx+1]; indy1 = [1:dwy:py-pwy+1];
lenx1 = length(indx1); leny1 = length(indy1);
for i = 1:lenx1
ind1 = [ind1,1+(py-pwy+1)*(indx1(i)-1):dwy:(py-pwy+1)+...
(py-pwy+1)*(indx1(i)-1)];
end
ind1 = ind1(2:end);
%obtain the partition blocks
partblock = parttmp(:,ind1); [pby,pbx] = size(partblock);
%create memory for output
tempres = zeros(dwy*dwx,blx);
tic
for count = 1:blx
%find missing elements
%make sure index is not empty vector(this happens if the observation window
%does not contain any elements
index = find(partblock(:,count) ==0); temp = partblock(index,count);
temp = temp-mean(temp);
%normalize
replicatetemp = temp*ones(1,cox);
distance = sum((replicatetemp-code(index,:)).^2,1);
[minvalue,minid] = min(distance);
%find index of missing elements
index1=find(block(:,count) ==0);
%select the proper correlation matrix
R=Rmatrix(index1,index1,minid);
%select the proper desired matrix
P=Pmatrix(index1,.,minid);
%obtain weights
w1 = R\P;
[ws1,ws2]=size(w1);
%weights are close to unity

```

```

%normalize the weights such that they sum to 1
w2=sum(w1,1); w3=(1-w2)./(ws1); w=w1+repmat(w3,ws1,1);
%process and the store the result
tempres(:,count) = ((w')*block(index1,count));
end
tblkp=toc
%form the output
out=col2im(tempres,[dwy,dwx],[szy,szx],'distinct');

```

BIBLIOGRAPHY

- [1] R. C. Hardie, K. J. Branard, J. G. Bogner, E. E. Armstrong, and E. A. Watson, "High-resolution image reconstruction from a sequence of rotated and translated frames and its application to an infrared imaging system," *Opt. Eng.*, vol. 37, no. 1, pp. 247–260, Jan 1998.
- [2] S. C. Park, M. K. Park, and M. G. Kang, "Super-resolution image reconstruction: A technical overview," *IEEE Signal Processing Mag.*, pp. 21–36, May 2003.
- [3] D. G. Sheppard, A. Bilgin, M. S. Nadar, B. R. Hunt, and M. W. Marcellin, "A vector quantizer for image restoration," *IEEE Trans. Image Processing*, vol. 7, pp. 119–124, Jan 1998.
- [4] K. E. Barner, A. M. Sarhan, and R. C. Hardie, "Partition-based weighted sum filters for image restoration," *IEEE Trans. Image Processing*, vol. 8, no. 5, pp. 740–745, May 1999.
- [5] M. Shao and K. E. Barner, "Optimization of partition-based weighted sum filters and their application to image denoising," *IEEE Trans. Image Processing*, Oct 2004, accepted for publication.
- [6] Y. Lin, R. C. Hardie, and K. E. Barner, "Subspace partition weighted sum filters for image restoration," *IEEE Signal Processing Letters*, Oct 2004, accepted for publication.
- [7] M. Shao, K. E. Barner, and R. C. Hardie, "Partition-based interpolation for image demosaicking and super-resolution reconstruction," *Opt. Eng.*, Jul 2004, accepted for publication.
- [8] R. A. Muse and R. C. Hardie, "A new non-uniformity correction technique based on readout architecture in focal plane arrays," in *The 6th World Multiconference on Systemics, Cybernetics and Informatics, Invited Session on Image Processing for Infrared Array Sensors: Nonuniformity Correction and Registration*, July 2002.

- [9] R. C. Hardie and M. M. Hayat, "A nonlinear-filter based approach to detector nonuniformity correction," in *Proceedings of the 2001 IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing*, June 2001.
- [10] P. M. Narendra and N. A. Foss, "Shutterless fixed pattern noise correction for infrared imaging arrays," *Proceedings of the SPIE-International Society of Optical Engineering, Technical Issues in Focal Plane Development*, vol. 282, pp. 44-51, 1981.
- [11] J. G. Harris, "Continuous-time calibration of vlsi sensors for gain and offset variations," *Proceedings of the SPIE International Symposium on Aerospace and Dual-Use Photonics, Smart Focal Plane Arrays and Focal Plane Array Testing*, M. Wigdor and M. A. Massie, eds., vol. 2474, pp. 23-33, 1995.
- [12] J. G. Harris and Y.-M. Chiang, "Nonuniformity correction using constant average statistics constraint: Analog and digital implementations," *Proceedings of the SPIE Aerospace/Defense Sensing and Controls, 1997, Infrared Technology and Applications XXIII*, B. F. Anderson and M. Strojnik, eds., vol. 3061, pp. 895-905, 1997.
- [13] Y.-M. Chiang and J. G. Harris, "An analog integrated circuit for continuous-time gain and offset calibration of sensor arrays," *J. Analog Integr. Circuits Signal Process.*, vol. 12, pp. 231-238, 1997.
- [14] W. F. O'Neil, "Dithered scan detector compensation," *Proceedings of the 1993 Meeting of the Infrared Information Symposium (IRIS) Specialty Group on Passive Sensors*, 1993.
- [15] R. Hardie, M. Hayat, E. Armstrong, and B. Yasuda, "Scene-based nonuniformity correction with video sequences and registration," *Applied Optics*, vol. 39, no. 8, pp. 1241-1250, Mar. 2000.
- [16] B. R. Ratliff and M. M. Hayat, "Algebraic scene-based nonuniformity correction in focal-plane arrays," *Proceedings of SPIE AereoSense'2001, in Infrared Imaging Systems: Design, Analysis, Modeling, and Testing XII*, vol. 4372, pp. 114-124, 2001.
- [17] M. M. Hayat, S. N. Torres, E. Armstrong, S. C. Cain, and B. Yasuda, "Statistical algorithm for nonuniformity correction in focal-plane arrays," *Applied Optics*, vol. 38, pp. 772-780, Mar. 1999.
- [18] S. N. Torres, E. B. Vera, and S. K. S. R. A. Reeves, "Adaptive scene-based non-uniformity correction method for infrared-focal plane arrays," *Proceedings of the SPIE Infrared Imaging Systems: Design, Analysis, Modelling, and Testing XIV*, vol. 5076, pp. 130-139, 2003.

- [19] R. Tsai and T. Huang, "Multiframe image restoration and registration," in *Advances in Computer Vision and Image Processing*, vol. 1, no. 5. Greenwich, CT: JAI Press Inc, May 1984, pp. 317-339.
- [20] S. Kim, N. Bose, and H. Valenzuela, "Recursive reconstruction of high resolution image from noisy undersampled multiframes," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, pp. 1013-1027, June 1990.
- [21] S. Kim and W. Su, "Recursive high-resolution reconstruction of image blurred multiframe images," *IEEE Trans. Image Processing*, vol. 2, pp. 534-539, Oct 1993.
- [22] M. Irani and S. Peleg, "Improving resolution by image registration," *CVGIP: Graph. Models Image Process.*, vol. 53, pp. 231-239, May 1991.
- [23] S. Mann and R. Picard, "Virtual bellows: construction of high quality stills from video," in *Proc. IEEE Int. Conf. Image Processing*, Austin, TX, Nov 1994.
- [24] B. Basile, A. Blake, and A. Zisserman, "Motion deblurring and super resolution from an image sequence," *EECV*, vol. 2, pp. 573-581, Apr 1996.
- [25] T. Connolly and R. Lane, "Gradient methods for superresolution," in *Proc. Int. Conf. Image Processing*, vol. 1, 1997, pp. 917-920.
- [26] R. Chan, T. Chan, M. Ng, W. Tang, and C. Wong, "Preconditioned iterative methods for high-resolution image reconstruction from multisensors," *Adv. Signal Process. Algorithms, Architectures, Implementations*, vol. 3461, pp. 348-357, 1998.
- [27] T. R. Tuinstra and R. C. Hardie, "High resolution image reconstruction from digital video by exploitation on non-global motion," *Opt. Eng.*, vol. 38, no. 5, May 1999.
- [28] N. Nguyen, P. Milanfar, and G. Golub, "A computationally efficient superresolution image reconstruction algorithm," *IEEE Trans. Image Processing*, vol. 10, no. 4, pp. 573-583, Apr 2001.
- [29] R. R. Schultz and R. L. Stevenson, "A bayesian approach to image extension for improved definition," *IEEE Trans. Image Processing*, vol. 3, no. 3, pp. 233-242, May 1994.
- [30] P. Cheeseman, B. Kanefsky, R. Kraft, J. Stutz, and R. Hanson, "Super-resolved surface reconstruction from multiple images," NASA Tech. Rep. FIA-94-12, Moffett Field, CA, Dec 1994.

- [31] R. R. Schultz and R. L. Stevenson, "Extraction of high-resolution frames from video sequences," *IEEE Trans. Image Processing*, vol. 5, pp. 996–1011, June 1996.
- [32] R. C. Hardie, K. J. Branard, and E. E. Armstrong, "Joint map registration and high-resolution image estimation using a sequence of undersampled images," *IEEE Trans. Image Processing*, vol. 6, no. 12, pp. 1621–1633, Dec 1997.
- [33] K. Sauer and J. Allebach, "Iterative reconstruction of band-limited images from non-uniformly spaced samples," *IEEE Trans. Circuits Syst.*, vol. CAS-34, pp. 1497–1505, 1987.
- [34] K. Aizawa, T. Komatsu, and T. Satio, "Acquisition of very high resolution images using stereo camera," in *Proc. SPIE Visual Communication and Image Processing*, vol. 1, Boston, MA, Nov 1991, pp. 318–328.
- [35] A. Tekalp, M. Ozkan, and M. Sezan, "High resolution image reconstruction from lower-resolution image sequences and space-varying image restoration," in *Proc. ICASSP '92*, vol. 3, San Francisco, CA, Mar 1992, pp. 169–172.
- [36] J. C. Gillette, T. M. Stadtmiller, and R. C. Hardie, "Reduction of aliasing in staring infrared imagers utilizing subpixel techniques," *Opt. Eng.*, vol. 34, no. 11, pp. 3130–3137, Nov 1995.
- [37] A. Patti, M. Sezan, and A. Tekalp, "Superresolution video reconstruction with arbitrary sampling lattices and nonzero aperture time," *IEEE Trans. Image Processing*, vol. 6, pp. 1064–1076, Aug 1997.
- [38] H. Shekarforoush and R. Chellappa, "Data-driven multi-channel super-resolution with application to video sequences," *J. Opt. Soc. Amer. A*, vol. 16, no. 3, pp. 481–492, Mar 1999.
- [39] N. Nguyen and P. Milanfar, "A wavelet-based interpolation restoration method for superresolution," *Circuits, Syst., Signal Process.*, vol. 19, no. 4, pp. 321–338, Aug 2000.
- [40] M. S. Alam, J. G. Bogner, R. C. Hardie, and B. J. Yasuda, "Infrared image registration using multiple translationally shifted aliased video frames," *IEEE Instrum. Meas. Mag.*, vol. 49, no. 5, Oct 2000.
- [41] F. M. Candocia and J. C. Principe, "Superresolution of images based on local correlations," *IEEE Trans. Neural Networks*, vol. 10, no. 2, pp. 372–380, Mar 1999.

- [42] S. Lertrattanapanich and N. K. Bose, "High resolution image formation from low resolution frames using delaunay triangulation," *IEEE Trans. Image Processing*, vol. 11, no. 12, pp. 1427–1441, Dec 2002.
- [43] A. C. Kokaram, R. D. Morris, W. J. Fitzgerald, and P. J. W. Rayner, "Interpolation of missing data in image sequences," *IEEE Trans. Image Processing*, vol. 4, no. 11, pp. 1509–1519, Nov 1995.
- [44] M. Shao, "Partition-based weighted sum filtering theory with application to image processing and engineering," Ph.D. dissertation, University of Delaware, 2004.
- [45] Y. Lin, "Partition-based image restoration," Ph.D. dissertation, University of Dayton, 2005.
- [46] B. D. Lucas and T. Kanade, "An iterative image registartion technique with an application to stereo vision," in *International Joint Conference on Artifical Intelligence*, Vancouver, Aug 1981.
- [47] 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.

R702032020