UNIVERSITY OF CALIFORNIA

SANTA CRUZ

LOCALLY ADAPTIVE KERNEL REGRESSION METHODS FOR MULTI-DIMENSIONAL SIGNAL PROCESSING

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRICAL ENGINEERING

by

Hiroyuki Takeda

September 2010

The Dissertation of Hiroyuki Takeda
is approved:

Professor Peyman Milanfar, Chair

Professor Benjamin Friedlander

Professor Roberto Manduchi

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Locally Adaptive Kernel Regression Methods for Multi-Dimensional Signal Processing

by

Hiroyuki Takeda

While digital imaging devices have been rapidly improved and widely used for consumer photography as well as microscope, medical, and astronomical imaging, the measured images/videos often suffer from some degradations, such as noise, blurring, aliasing effects, and more due to the electromechanical limitations. Although many different methods have been proposed, they are often designed for recovering the images from one specific degradation.

In this work, we study the *kernel regression* (KR) as a general restoration approach for multidimensional signals. The classic KR is a statistical technique that enables us to regard a variety of image/video restoration problems as *regression*, and it has a few advantageous properties: (i) the classic KR is a spatially-adaptive point estimation procedure that is capable of finding missing pixels and smoothing noise-ridden pixels, and (ii) it requires few assumptions on the underlying multidimensional data. Furthermore, the data-adaptive KR we propose locally learns the optimal filter coefficient from not only the spatial density of the given data but also the radiometric structures of the underlying data. Thus, it is applicable to a wide variety of problems, such as denoising, interpolation, deblurring and super-resolution tasks for images/videos. The experimental results with synthetic/real data presented in each chapter will show the superiority of the KR approach.

To my family.

Acknowledgments

First, I wish to express my gratitude to my advisor, Professor Peyman Milanfar. Without his support and guidance, I would have never gotten to where I am now. Since I joined the MDSP group as a master's student in 2004, he has taught me not only the basics of the image processing but also a number of practical skills to accomplish the projects. In addition, his advice and suggestions helped me develop analytical/creative thinking. I consider myself lucky that I had an opportunity to work with him throughout my research.

I am indebted to Professor Benjamin Friedlander and Professor Roberto Manduchi for being the thesis committee members, and Dr. Peter van Beek for being my mentor during the summer internship at Sharp in 2007 and 2008. Also, I would like to thank Professor Michael Elad and Professor Sina Farsiu for providing the valuable feedback to my journal papers.

I thank the former students of Professor Milanfar, Dr. Dirk Robinson, and Dr. Morteza Shaharam, and Dr. Amyn Poonawara, and the new students, Priyam, Hae Jong, and Xiang. They are excellent researchers to have technical discussions as well as nice friends to hang out. My life in Santa Cruz was enjoyable because of them.

I am thankful for the best friend, Dr. Shigeru (CJ) Suzuki. Without him, I would not be able to survive in graduate school. He always gives me a unique question which indirectly teaches me how to see the things from many other perspectives.

Finally, I want to thank my parents, Ren Jie Dong and Akemi Takeda, my sisters, Asako and Tomoko, my grand mother, Kinu, and my wife, Mayumi, for their consideration. I dedicate this work to my family.

Santa Cruz, California

June 7th, 2010

Hiroyuki Takeda

Chapter 1

Introduction

Abstract— This chapter addresses the *regression problem* in which we study and understand the typical degradation models of imaging system, such as downsampling, blurring, and noise effects. In addition, we consider the given samples (pixels) measured irregularly spaced, and hence, the data model becomes quite general. Instead of restoring/reconstructing images while making specific assumptions on the given data, we introduce *classic kernel regression*, which requires minimal assumptions, as a fundamental tool for the regression problem.

1.1 Regression Problem

The ease of use and cost efficiency have contributed to the growing popularity of digital imaging systems. However, digital imaging systems always suffer from some sort of degradations. For instance, inferior spatial resolution causes the apparent aliasing effect due to the limited number of CCD pixels. In this case, an proper interpolation is necessary to reduce the aliasing and increase the number of pixels. On

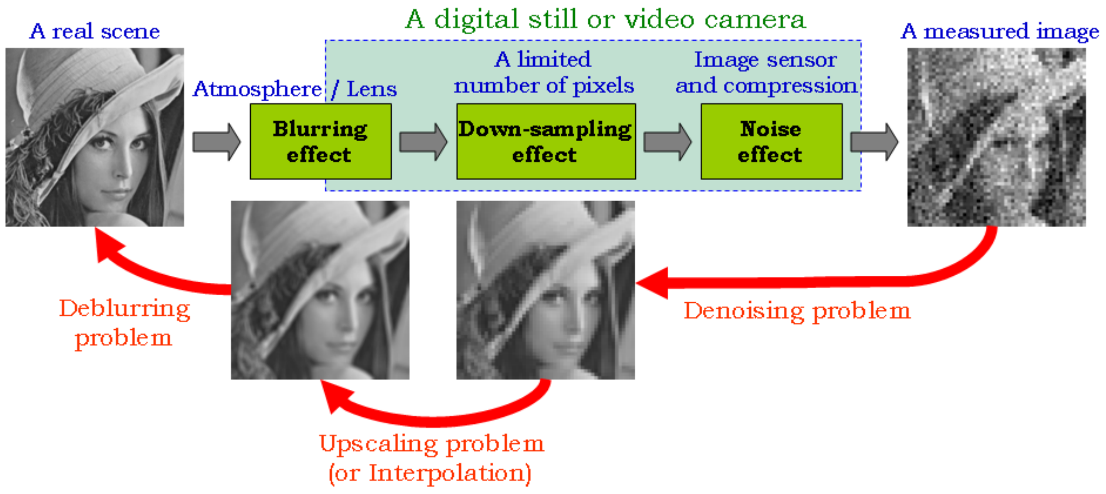


Figure 1.1: A simple imaging system model of commercial digital image/video cameras.

the other hand, using denser array CCD (with smaller pixels) increases not only the spatial image resolution but also results in noisier images since the number of photons for each pixel is insufficient. Longer exposure time allows each pixel to obtain more photons and suppresses the noise effect, then the captured images may be degraded by motion blur effect because of camera shaking or object movements. In addition, out-of-focus also blurs images. Hence, denoising and deblurring are necessary. As a cost efficient way, image processing methods have even exploited through the years to improve the quality of digital images. This thesis focuses on regression problems that attempt to recover the noiseless high-frequency information corrupted by the limitations of imaging system, as well as the degradation processes such as compression. More specifically, the regression (estimation) problem in this thesis includes denoising, interpolating (and upscaling), and deblurring. Figure **1.1** illustrates a simple imaging system, and it shows how the measured images are degraded at each step. When we take a gray-scale photo of a real scene (Lena's face), first the scene is blurred by atmo-

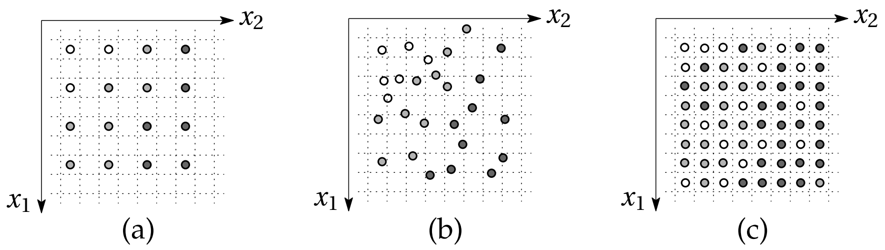


Figure 1.2: Possible spatial data sampling models: (a) an incomplete set of regularly sampled data, (b) irregularly sampled data set, and (c) an complete set of regularly sampled data.

sphere and camera lenses, and the camera sensor measures the blurred scene. Due to the limited number of pixels on the camera sensor, the measured image has a finite number of pixels, and this is referred to as discretization or the downsampling effect. Also due to the physical shortcoming of camera sensor (such as thermal noise and dark current), the measured pixels are contaminated by some noise. Finally, cameras usually compress the image in for example JPEG format before saving it, and hence the image is also degraded by compression artifacts (e.g., blocking artifacts).

In this thesis, we study regression, as a tool not only for interpolation of regularly sampled data, but also for restoration and enhancement of noisy and possibly irregularly sampled data. Figure **1.2**(a) illustrates an example of the former case, where we opt to upsample an image by a factor of $1:2$ in each direction as well as to remove noise. Figure **1.2**(b) illustrates an example of the latter case, where an irregularly sampled noisy data is to be reconstructed onto a high resolution regular grid. Besides inpainting applications [1] and reconstruction of irregularly sampled image data is essential for applications such as multi-frame super-resolution, where several low-resolution images are fused (interlaced) onto a high-resolution grid [2]. Figure **1.3** represents a diagram representation of such super-resolution algorithm. We note that "denoising" is a special case of the regression problem where samples at all desired

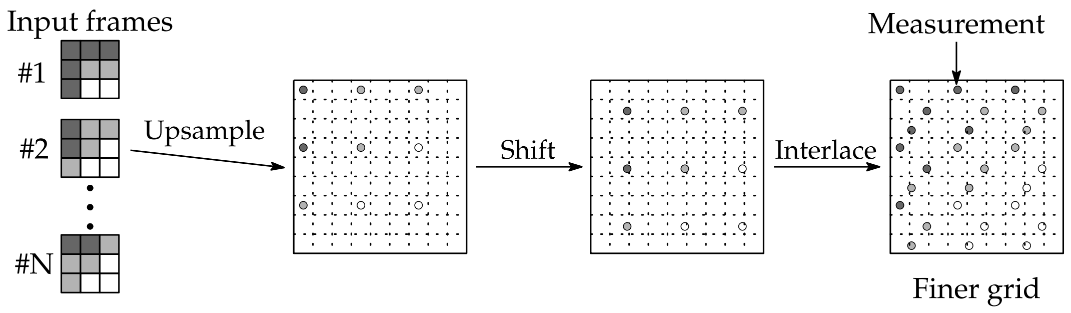


Figure 1.3: Image fusion often yields us irregularly sampled data.

pixel locations are given as illustrated in Figure **1.2**(c), but these samples are corrupted, and need to be restored.

1.2 Contributions

As introduced above, the regression problem covers a variety of signal (image/ video) processing problems, i.e. interpolation, denoising, and deblurring. Furthermore, as illustrated in Figures **1.2** and **1.3**, the given samples could be incomplete and even irregularly spaced, and thus, few assumptions are made on the underlying signals. The main contribution of this thesis is to describe and propose the *kernel regression* framework as an effective tool even for irregularly spaced samples. Kernel regression is known as a nonparametric approach that requires minimal assumptions, and hence the framework is one of the suitable approaches to the regression problem. In the rest of this introductory chapter, we study the classic kernel regression framework, and expand it and develop applications for image and video in the following chapters. Specifically, this thesis is structured as follows:

▷ **Chapter 2 – Data Adaptive Kernel Regression**

Having introduced the classic kernel regression in Chapter 1, in this chapter, we propose a novel adaptive generalization of kernel regression with excellent results in both denoising and interpolation for single and multi-frame applications.

▷ **Chapter 3 – Kernel-Based Image Deblurring**

Using the knowledge of the data-adaptive kernel regression, we apply it for image deblurring. Deblurring is a very challenging problem and we present a regularization based on the data-adaptive method, which reduces both noise and ringing effectively, with experimental results.

▷ **Chapter 4 – Multi-Dimensional Kernel Regression for Video Upscaling**

In this chapter, introducing the 3rd dimension (time axis), we develop two types of spatiotemporal application for video denoising and upscaling based on the data-adaptive kernel regression. One explicitly takes local motion information into account. The other approach captures local signal structures in 3-D, which contain local motion information implicitly, and it avoids explicit subpixel motion estimation.

▷ **Chapter 5 – Video Deblurring**

Upscaled videos in space and time often suffer from blurring effects, motion (temporal) blur as well as spatial blur. In this chapter, we discuss motion (temporal) deblurring in particular along with the necessity of temporal upscaling of video (so-called *frame-rate upconversion*).

Finally, in Chapter 6, we conclude the thesis and discuss possible topics of future work and its applications presented in this thesis.

1.3 Classic Kernel Regression and its Properties

In this section, we review the classic kernel regression framework, provide some intuitions on computational efficiency as well as weaknesses of this basic approach, and motivate the development of more powerful regression tools presented in the following chapters.

1.3.1 Kernel Regression in 1-D

Classical parametric image processing methods rely on a specific model of the signal of interest, and seek to compute the parameters of this model in the presence of noise. Examples of this approach are presented in diverse problems ranging from denoising to upscaling, interpolation, and deblurring. A generative model based upon the estimated parameters is then produced as the best estimate of the underlying signal.

In contrast to the parametric methods, non-parametric methods rely on the data itself to dictate the structure of the model, in which case this implicit model is referred to as *a regression function* [3]. With the relatively recent emergence of machine learning methods, kernel methods have become well-known and used frequently for pattern detection and discrimination problems [4]. Surprisingly, it appears that the corresponding ideas in non-parametric estimation – what we call here *kernel regression*, are not widely recognized or used in the image and video processing literature. Indeed, in the last decades, several concepts related to the general theory we promote here have been rediscovered in different guises, and presented under different names such as *normalized convolution* [5, 6], *bilateral filter* [7, 8], *edge-directed interpolation* [9], and *moving least-squares* [10]. Later in this chapter, we shall say more about some of these concepts and their relation to the general regression theory. To simplify this introductory presentation, we threat the 1-D case where the measured data are given by

$$y_i = z(x_i) + \varepsilon_i, \quad i = 1, 2, \cdots, P, \tag{1.1}$$

where $z(\cdot)$ is the (hitherto unspecified) regression function, ε_i is the independent and identically distributed zero mean noise value (with otherwise no particular statistical distribution assumed), and P is the number of measured samples in a local analysis window. As such, kernel regression provides a rich mechanism for computing point-wise estimates of the function with minimal assumptions about global signal or noise models.

While the specific form of the regression function $z(\cdot)$ may remain unspecified, if we assume that it is locally smooth to some order N, then in order to estimate the value of the function at any point x given the data, we can rely on a generic local expansion of the function about this point. Specifically, if the position of interest x is near the sample at x_i, we have the N-term Taylor series

$$\begin{aligned} z(x_i) &= z(x) + z'(x)(x_i - x) + \frac{1}{2!} z''(x)(x_i - x)^2 + \cdots + \frac{1}{N!} z^{(N)}(x)(x_i - x)^N \\ &= \beta_0 + \beta_1(x_i - x) + \beta_2(x_i - x)^2 + \cdots + \beta_N(x_i - x)^N, \end{aligned} \tag{1.2}$$

where $z'(\cdot)$ and $z^{(N)}(\cdot)$ are the first and N-th derivatives of the regression function. The above suggest that if we now think of Taylor series as a local representation of the regression function, estimating the parameters β_0 can yield the desired (local) estimate of the regression function based on the data[1]. Indeed, the parameters $\{\beta_n\}_{n=1}^{N}$ will provide localized information on the n-th derivatives of the regression function. Naturally,

[1]Indeed the local approximation can be build upon bases other than polynomials [3].

since this approach is based on *local* approximation, a logical step to take is to estimate the parameters $\{\beta_n\}_{n=0}^{N}$ from the data while giving the nearby samples higher weight than samples farther away. A least-squares formulation capturing this idea is to solve the following optimization problem:

$$\min_{\beta_n} \sum_{i=1}^{P} \left[y_i - \beta_0 - \beta_1 (x_i - x) - \beta_2 (x_i - x)^2 - \cdots - \beta_N (x_i - x)^N \right]^2 \frac{1}{h} K\left(\frac{x_i - x}{h} \right), \tag{1.3}$$

where $K(\cdot)$ is the *kernel function* which penalizes distant away from the local position where the approximation is centered, ant the *smoothing parameter* h (also called the "bandwidth") controls the strength of this penalty [3]. In particular, the function K is a symmetric function which attains its maximum at zero, satisfying

$$\int \delta K(\delta) d\delta = 0, \quad \int \delta^2 K(\delta) d\delta = c, \tag{1.4}$$

where c is some constant value[2]. The choice of the particular form of the function K is open, and may be selected as a Gaussian, exponential, or other forms which comply with the above constraints. It is known that for the case of classic regression the choice of the kernel has only a small effect on the accuracy of estimation [11] and therefore preference is given to the differentiable kernels with low computational complexity such as the Gaussian kernel. The effect of kernel choice for the data-adaptive algorithms, presented in Chapter 2, is an interesting avenue of research, which is outside the scope of this thesis and part of our future work discussed in Section 6.2.1.

Several important points are worth making here. First, the above structure allows for tailoring the estimation problem to the *local* characteristics of the data, whereas

[2]Basically, the only conditions needed for the regression framework are that the kernel function be non-negative, symmetric and unimodal. For instance, unlike the kernel *density* estimation problems [11], even if the kernel weights in the minimization (1.3) do not sum up to one, they will be normalized in the estimator derived from the minimization (see (1.34) and Appendix A).

the standard parametric model is generally intended as a more global fit. Second, in the estimation of the local structure, higher weight is given to the nearby data as compared to samples that are farther away from the center of the analysis window. Meanwhile, this approach does not specifically require the data to follow a regular or equally spaced sampling structure. More specifically, so long as the samples are near the point x the framework is valid. Again this is in contrast to the general parametric approach which generally either does not directly take the location of the data samples into account, or relies on regular sampling over a grid. Third, and no less important as we will describe in Chapter 2, the data-adaptive approach is both useful for *denoising*, and equally viable for *interpolation* of sampled data at points where no actual samples exist. Given the above observations, the kernel-based methods are well-suited for a wide class of image/video processing problems of practical interest.

Returning to the estimation problem based upon (1.3), one can choose the order N to effect an increasingly more complex local approximation of the signal. In the non-parametric statistics literature, locally constant, linear, and quadratic approximation (corresponding to $N = 0, 1, 2$) have been considered most widely [3, 12, 13, 14]. In particular, choosing $N = 0$, a locally *linear* filter is obtained, which is known as the *Nadaraya-Watson* Estimator (NWE) [15]. Specifically, this estimator has the form:

$$\hat{z}(x) = \frac{\sum_{i=1}^{P} K_h(x_i - x)\, y_i}{\sum_{i=1}^{P} K_h(x_i - x)}, \tag{1.5}$$

where

$$K_h(x_i - x) = \frac{1}{h} K\left(\frac{x_i - x}{h}\right). \tag{1.6}$$

The NWE is the simplest manifestation of an adaptive filter resulting from kernel regression framework. As we shall see later in Chapter 2, the *bilateral filter* [7, 8] can be interpreted as a generalization of NWE with a modified kernel definition.

Of course, higher order approximations ($N > 0$) are also possible. The choice of order in parallel with the smoothness (h) affects the bias and variance of the estimate. Mathematical expression for bias and variance can be found in [16, 17], and therefore here we only briefly review their properties. In general, lower order approximates, such as NWE, result in smoother signals (large bias and small variance) as there are fewer degrees of freedom. On the other hand over-fitting happens in regressions using higher orders of approximation, resulting in small bias and large estimation variance. We illustrate the effect of the regression orders from $N = 0$ to 2 with a fixed smoothing parameter in Figure **1.4**, where the blue curve is the regression (true) function, the gray circles (y_1, y_2, and y_3) are the measurements with some noise at x_1, x_2 and x_3, respectively, and, in this illustration, we estimate an unknown value at the position of interest x located between the samples y_2 and y_3 with different regression orders. First, for the zeroth order (constant model, $N = 0$), we take only the constant term of Taylor series into account, i.e. β_0. In this case, the kernel regression estimates $z(x)$ by NWE (1.34); a weighted average of nearby samples with the weights that the kernel function gives. The red circle is the estimated value and the red line is the estimated fitted line. Similar to the zeroth order case, we draw the estimate values and fitted lines by the first and second order kernel regression, which incorporate up to the linear and quadratic terms, respectively. As seen in Figures **1.4**(a)-(c), the estimated line fits the neighboring samples better the higher the regression order becomes. This is because a high order regressor has more degrees of freedom, which is the cause of small bias and large variance. We also note that smaller values for h result in small bias and

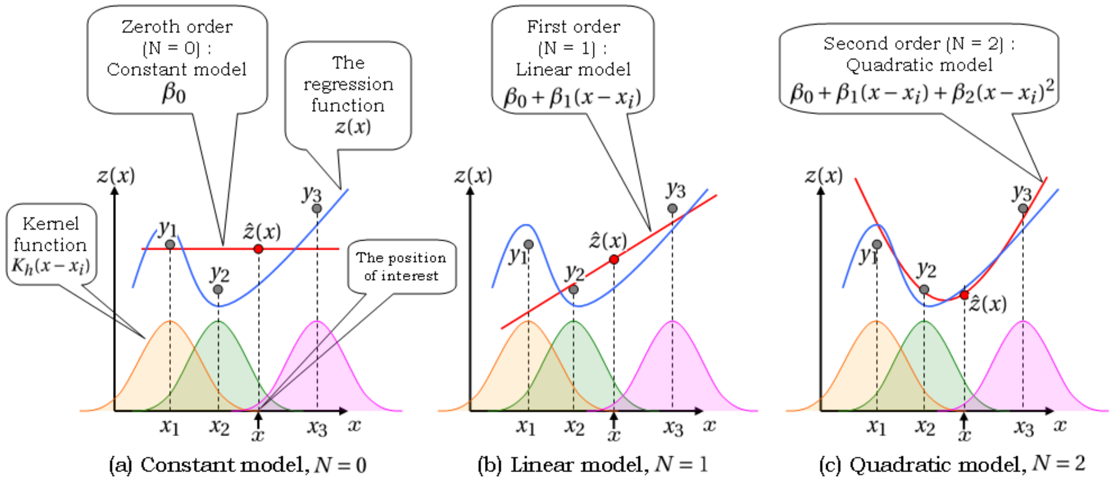

Zeroth order (N = 0) : Constant model β_0
The regression function $z(x)$
Kernel function $K_h(x-x_i)$
The position of interest
$z(x)$
y_1
y_2
y_3
$\hat{z}(x)$
x_1 x_2 x x_3 x
(a) Constant model, $N = 0$
First order (N = 1) : Linear model $\beta_0 + \beta_1(x - x_i)$
(b) Linear model, $N = 1$
Second order (N = 2) : Quadratic model $\beta_0 + \beta_1(x - x_i) + \beta_2(x - x_i)^2$
(c) Quadratic model, $N = 2$

Figure 1.4: The effect of the regression orders: (a) Zeroth order kernel regression (constant model, $N = 0$), (b) First order kernel regression (linear model, $N = 1$), and (c) Second order kernel regression (quadratic model, $N = 2$).

consequently large variance in estimates. Optimal order and smoothing parameter selection procedures are studied in [10].

The performance of kernel regressors of different orders is compared in the illustrative example of Figure **1.5**. In the first experiment, illustrated in the first row, a set of moderately noisy[3] regularly sampled data are used to estimate the underlying function. As expected, the computationally more complex high order interpolation ($N = 2$) results in a better estimate than the lower ordered interpolators ($N = 0$ or $N = 1$). The presented quantitative comparison of the Peak Signal to Noise Ratio[4] (PSNR) supports this claim. The second experiment, illustrated in the second row, shows that for the heavily noisy data sets (variance of the additive Gaussian noise 0.5), the performance of lower order regressors is better. Note that the performance of the zeroth and first

[3]Variance of the additive Gaussian noise is 0.1. Smoothing parameter is chosen by the cross validation method (Section 1.3.4).

[4]$\text{PSNR} = 20\log_{10}\left(\frac{\text{Peak Signal Value}}{\text{Root Mean Square Error}}\right)$ [dB]

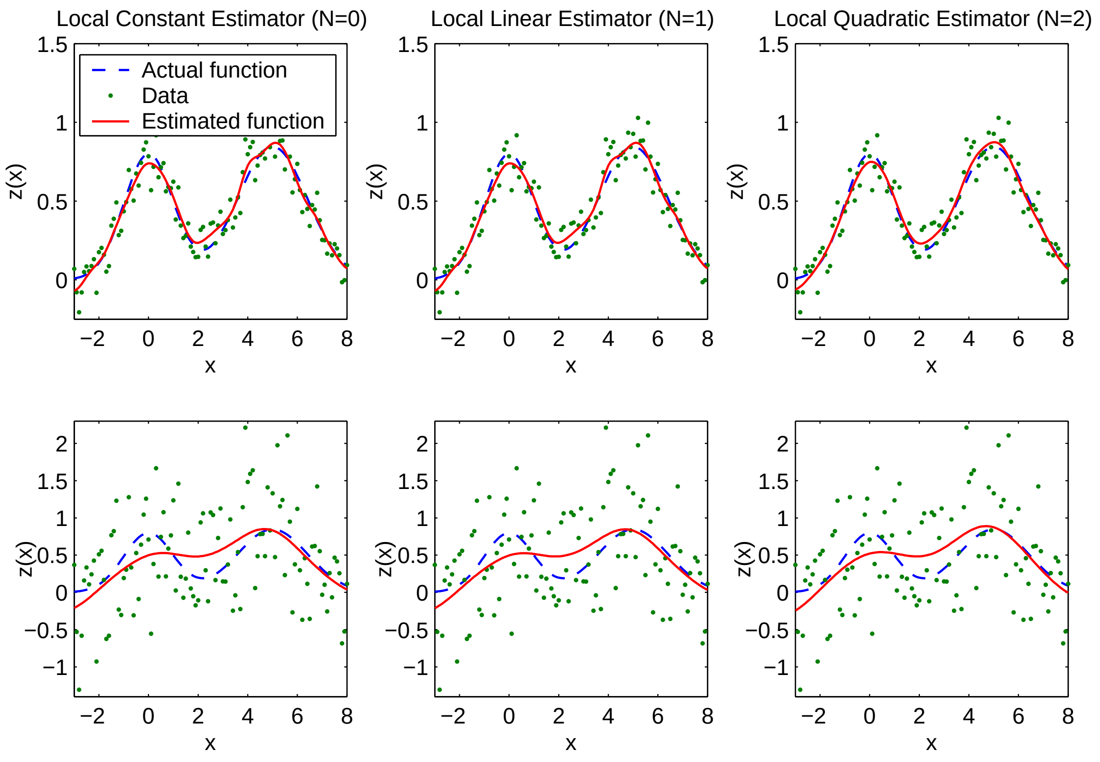


Figure 1.5: Examples of local polynomial regression on an equally-spaced data set. The signals in the first and second rows are contaminated with Gaussian noise of SNR = 9[dB] and −6.5[dB], respectively. The dashed, solid lines, and dots represent the actual function, estimated , and the noisy data, respectively. The columns from left to right show the constant, linear, and quadratic interpolation results. Corresponding PSNR values for the first row experiments are 28.78, 28.78, 30.26[dB] and for the second row are as 15.41, 15.41, 15.37[dB].

order estimators ($N = 0$ and 1) for these equally-spaced samples experiments are identical. In Section 1.3.3, we study this property in more details for the 2-D case.

1.3.2 Related Regression Methods

In addition to kernel regression methods which we are advocating, there are several other related and effective regression methods such as B-spline interpolation [18], orthogonal series [13, 19], cubic spline interpolation [20], and spline smoother [13, 18, 21]. We briefly review some or these methods in this section.

In orthogonal series methods, instead of using Taylor series, the regression function z can be represented by a linear combination of other basis functions, such as Legendre polynomials, wavelet bases, Hermite polynomials [10] and so on. In the 1-D case, such a model in general is represented as

$$z(x) = \sum_{n=0}^{N} \beta_n \phi_n(x). \tag{1.7}$$

The coefficients $\{\beta_n\}_{n=0}^{N}$ are the unknown parameters we want to estimate. We refer the interested reader to [13] (pages 104-107) which offers further examples and insights.

Following the notation used in the previous subsection, the B-spline regression is expressed as the linear combination of shifted spline functions $B^n(\cdot)$:

$$z(x) = \sum_{k} \beta_k B^n(x-k), \tag{1.8}$$

where the n-th order B-spline function is defined as a $n+1$ times convolution of the zeroth order B-spline [18], that is

$$B^n(x) = \underbrace{B^0(x) * B^0(x) * \cdots * B^0(x)}_{n+1}, \tag{1.9}$$

where

$$B^0(x) = \begin{cases} 1, & -\frac{1}{2} < x < \frac{1}{2} \\ \frac{1}{2}, & |x| = \frac{1}{2} \\ 0, & \text{else} \end{cases} . \tag{1.10}$$

The scalar k in (1.7), often referred to as the *knot*, defines the center of the spline. Least-squares is usually exploited to estimate the B-spline coefficients $\{\beta_k\}$.

The B-spline interpolation method bears some similarities to the kernel regression method. One major difference between these methods is the number and position of the knots as illustrated in Figure **1.6**. While in the classic B-spline method the

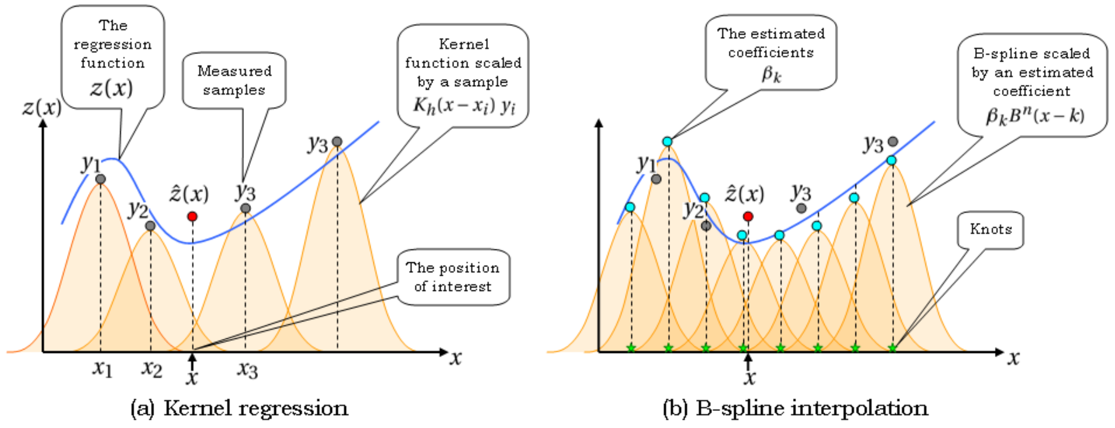

The regression function $z(x)$
Measured samples
Kernel function scaled by a sample $K_h(x-x_i)\,y_i$
$z(x)$
y_1
y_2
$\hat{z}(x)$
y_3
y_3
The position of interest
x_1
x_2
x
x_3
x
(a) Kernel regression
The estimated coefficients β_k
B-spline scaled by an estimated coefficient $\beta_k B^n(x-k)$
y_1
y_2
$\hat{z}(x)$
y_3
y_3
Knots
x
x
(b) B-spline interpolation

Figure 1.6: A comparison of the position of knots in (a) kernel regression and (b) classic B-spline method.

knots are located in equally spaced position, in the case of kernel regression, the knots are implicitly located on the sample positions. A related method, the *Non-Uniform Rational B-Spline* is also proposed to address this shortcoming of the classic B-spline method, by irregularly positioning the knots with respect to the underlying signals [22].

Cubic spline interpolation technique is one of the most popular members of the spline interpolation family which is based on fitting a polynomial between any pair of consecutive data. Assuming that the second derivative of the regression function exists, cubic spline interpolator is defined as

$$z(x) = \beta_0(i) + \beta_1(i)(x_i - x) + \beta_2(i)(x_i - x)^2 + \beta_3(i)(x_i - x)^3, \quad x \in [x_i, x_{i+1}], \tag{1.11}$$

where, under following boundary conditions

$$\begin{gathered} z(x)\Big|_{x=-x_i} = z(x)\Big|_{x=+x_i}, \quad z'(x)\Big|_{x=-x_i} = z'(x)\Big|_{x=+x_i}, \\ z''(x)\Big|_{x=-x_i} = z''(x)\Big|_{x=+x_i}, \quad z''(x_1) = z''(x_p) = 0, \end{gathered} \tag{1.12}$$

all the coefficients $\beta_n(i)$ can be uniquely defined [20].

Note that an estimated curve by cubic spline interpolation passes through all the given data points which is ideal for the noiseless data case. However, in most practical applications, data are contaminated with noise and therefore such perfect fits are no longer desirable. Consequently a related method called spline smoother has been proposed [18]. In spline smoothing method the above hard conditions are replaced with soft ones, by introducing them as Bayesian priors which penalize rather than constrain non-smoothness in the interpolated signals. A popular implementation of the spline smoother [18] is given by

$$\hat{z}(x) = \arg\min_{z(x)} \left[\sum_{i=1}^{P} \left\{ y_i - z(x_i) \right\}^2 + \lambda \left\| z'' \right\|_2^2 \right], \tag{1.13}$$

where

$$\left\| z'' \right\|_2^2 = \int \{ z''(x) \}^2 dx, \tag{1.14}$$

and $z(x_i)$ can be replaced by either (1.7) or any orthogonal series (e.g. [23]), and λ is the regularization parameter. Note that assuming a continuous sample density function, the solution to this minimization problem is equivalent to NWE (1.34), with the following kernel function and smoothing parameter

$$K(\delta) = \frac{1}{2} \exp\left(-\frac{|\delta|}{\sqrt{2}} \right) \sin\left(\frac{|\delta|}{\sqrt{2} + \frac{\pi}{4}} \right), \tag{1.15}$$

$$h(x_i) = \sqrt[4]{\frac{\lambda}{P f(x_i)}}, \tag{1.16}$$

where f if the density of sample [13, 11]. Therefore, spline smoother is a special form of kernel regression.

In Chapter 2, we compare the performance of the spline smoother with the data-adaptive kernel regression method, and we give some intuitions for the outstanding performance of the kernel regression methods.

Li *et al.* proposed an edge-directed interpolation method for upscaling regularly sampled images [9]. The interpolation is implemented by weighted average of the four immediate neighboring pixels in a regular upsampling scenario where the filter coefficients are estimated using the classic covariance estimation method [24].

The normalized convolution method presented in [5] is theoretically identical to the classic kernel regression method (considering a different basis function), which we show a simplified version of the adaptive kernel regression introduced in Chapter 2. An edge adaptive version of this work is also proposed in [6].

We note that other popular edge adaptive denoising or interpolation technique are available in the literature, such as the PDE based methods [25, 26, 27]. A denoising experiment using the anisotropic diffusion (the second order PDE) method of [25] is presented in Chapter 2, however we refrain from a detailed discussion and comparison of all these diverse methods.

1.3.3 Kernel Regression in 2-D

Similar to the 1-D case in (1.1), the data model in 2-D is given by

$$y_i = z(\mathbf{x}_i) + \varepsilon_i, \quad i = 1, \cdots, P, \quad \mathbf{x}_i = [x_{1i}, x_{2i}]^T, \tag{1.17}$$

where y_i is a noisy sample at a sampling position $\mathbf{x}_i$ (Note: x_{1i} and x_{2i} are spatial coordinates), $z(\cdot)$ is again the (unspecified and bivariate) *regression function* to be estimated, ε_i is an i.i.d. zero mean noise, and P is the total number of samples in an arbitrary "window" around a position $\mathbf{x}$ of interest as shown in Figure **1.7**. Correspondingly, the local representation of the regression function using Taylor series (up to the N-th

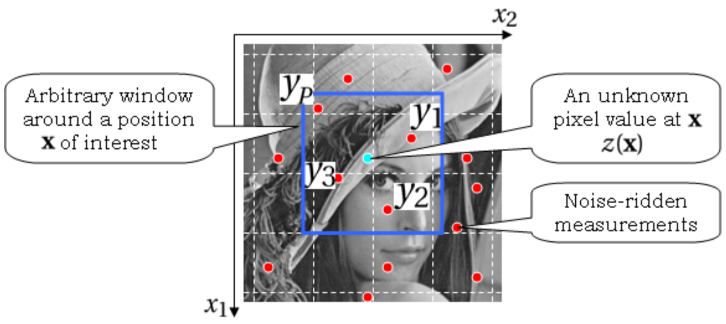

x_2
x_1
Arbitrary window around a position $\mathbf{x}$ of interest
An unknown pixel value at $\mathbf{x}$ $z(\mathbf{x})$
Noise-ridden measurements
y_P
y_1
y_2
y_3

Figure 1.7: The data model for the kernel regression framework in 2-D.

polynomial) is given by

$$\begin{aligned} z(\mathbf{x}_i) &\approx z(\mathbf{x}) + \{\nabla z(\mathbf{x})\}^T(\mathbf{x}_i - \mathbf{x}) + \frac{1}{2}(\mathbf{x}_i - \mathbf{x})^T\{\mathcal{H}z(\mathbf{x})\}(\mathbf{x}_i - \mathbf{x}) + \cdots \\ &= z(\mathbf{x}) + \{\nabla z(\mathbf{x})\}^T(\mathbf{x}_i - \mathbf{x}) + \frac{1}{2}\text{vec}^T\{\mathcal{H}z(\mathbf{x})\}\text{vec}\left\{(\mathbf{x}_i - \mathbf{x})(\mathbf{x}_i - \mathbf{x})^T\right\} + \cdots, \end{aligned} \tag{1.18}$$

where ∇ and $\mathcal{H}$ are the gradient (2×1) and Hessian (2×2) operators, respectively, and vec$(\cdot)$ is the vectorization operator, which lexicographically orders a matrix into a column-stack vector. Defining vech$(\cdot)$ as the half-vectorization operator of the "lower-triangular" portion of a symmetric matrix, e.g.,

$$\begin{aligned} \text{vech}\left(\begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix}\right) &= \begin{bmatrix} a_{11} & a_{12} & a_{22} \end{bmatrix}^T \\ \text{vech}\left(\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}\right) &= \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{22} & a_{23} & a_{33} \end{bmatrix}^T \end{aligned} \tag{1.19}$$

and considering the symmetry of the Hessian matrix, the local representation in (1.18) is simplified to

$$z(\mathbf{x}_i) \approx \beta_0 + \boldsymbol{\beta}_1^T(\mathbf{x}_i - \mathbf{x}) + \boldsymbol{\beta}_2^T\text{vech}\left\{(\mathbf{x}_i - \mathbf{x})(\mathbf{x}_i - \mathbf{x})^T\right\} + \cdots. \tag{1.20}$$

then, comparison of (1.18) and (1.20) suggests that β_0 is the pixel value of interest, and the vectors $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$ are the first and second derivatives, respectively, i.e.,

$$\begin{aligned}
\beta_0 &= z(\mathbf{x}), && (1.21)\\
\boldsymbol{\beta}_1 &= \nabla z(\mathbf{x}) = \left[\frac{\partial z(\mathbf{x})}{\partial x_1} \quad \frac{\partial z(\mathbf{x})}{\partial x_2}\right]^T, && (1.22)\\
\boldsymbol{\beta}_2 &= \frac{1}{2}\left[\frac{\partial^2 z(\mathbf{x})}{\partial x_1^2} \quad 2\frac{\partial^2 z(\mathbf{x})}{\partial x_1 \partial x_2} \quad \frac{\partial^2 z(\mathbf{x})}{\partial x_2^2}\right]^T. && (1.23)
\end{aligned}$$

As in the case of univariate data, the $\boldsymbol{\beta}_n$'s are computed from the following optimization problem:

$$\min_{\{\boldsymbol{\beta}_n\}} \sum_{i=1}^{P} \left[y_i - \beta_0 - \boldsymbol{\beta}_1^T(\mathbf{x}_i - \mathbf{x}) - \boldsymbol{\beta}_2^T \mathrm{vech}\{(\mathbf{x}_i - \mathbf{x})(\mathbf{x}_i - \mathbf{x})^T\} - \cdots\right]^2 K_{\mathbf{H}}(\mathbf{x}_i - \mathbf{x}), \tag{1.24}$$

with

$$K_{\mathbf{H}}(\mathbf{x}_i - \mathbf{x}) = \frac{1}{\det(\mathbf{H})} K(\mathbf{H}^{-1}(\mathbf{x}_i - \mathbf{x})), \tag{1.25}$$

where K is the 2-D realization of the kernel function, and $\mathbf{H}$ is the 2×2 smoothing matrix, which will be studied more carefully in Chapter 2. For example, if we choose Gaussian function for K, the kernel function is expressed as

$$K_{\mathbf{H}}(\mathbf{x}_i - \mathbf{x}) = \frac{1}{2\pi\sqrt{\det(\mathbf{H}^T\mathbf{H})}} \exp\left\{-\frac{1}{2}(\mathbf{x}_i - \mathbf{x})^T \left(\mathbf{H}^T\mathbf{H}\right)^{-1} (\mathbf{x}_i - \mathbf{x})\right\}. \tag{1.26}$$

Regardless of the regression order (N) and the dimensionality of the regression function, we can rewrite the optimization problem (1.24) as a weighted least squares optimization problem [3, 10, 28]:

$$\widehat{\mathbf{b}} = \arg\min_{\mathbf{b}} \left[(\mathbf{y} - \mathbf{X}\mathbf{b})^T \mathbf{K}(\mathbf{y} - \mathbf{X}\mathbf{b})\right], \tag{1.27}$$

where

$$
\begin{aligned}
\mathbf{y} &= \left[y_1 \;\; y_2 \;\; \cdots \;\; y_P\right]^T, && (1.28)\\
\mathbf{b} &= \left[\beta_0 \;\; \boldsymbol{\beta}_1^T \;\; \cdots \;\; \boldsymbol{\beta}_N^T\right]^T, && (1.29)\\
\mathbf{K} &= \mathrm{diag}\Big[K_{\mathbf{H}}(\mathbf{x}_1-\mathbf{x}) \;\; K_{\mathbf{H}}(\mathbf{x}_2-\mathbf{x}) \;\; \cdots \;\; K_{\mathbf{H}}(\mathbf{x}_P-\mathbf{x})\Big], && (1.30)\\
\mathbf{X} &= \begin{bmatrix} 1 & (\mathbf{x}_1-\mathbf{x})^T & \mathrm{vech}^T\{(\mathbf{x}_1-\mathbf{x})(\mathbf{x}_1-\mathbf{x})^T\} & \cdots \\ 1 & (\mathbf{x}_2-\mathbf{x})^T & \mathrm{vech}^T\{(\mathbf{x}_2-\mathbf{x})(\mathbf{x}_2-\mathbf{x})^T\} & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ 1 & (\mathbf{x}_P-\mathbf{x})^T & \mathrm{vech}^T\{(\mathbf{x}_P-\mathbf{x})(\mathbf{x}_P-\mathbf{x})^T\} & \cdots \end{bmatrix} && (1.31)
\end{aligned}
$$

with "diag" defining a diagonal matrix. Using the notation above, the optimization (1.24) provides the weighted least square estimator:

$$\widehat{\mathbf{b}} = \left(\mathbf{X}^T\mathbf{K}\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{K}\,\mathbf{y}. \tag{1.32}$$

Since our primary interest is to compute an estimate of the image (pixel values), the necessary computations are limited to the ones that estimate the parameter β_0. Therefore, the estimator is simplified to

$$\hat{z}(\mathbf{x}) = \hat{\beta}_0 = \mathbf{e}_1^T\left(\mathbf{X}^T\mathbf{K}\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{K}\,\mathbf{y}, \tag{1.33}$$

where $\mathbf{e}_1$ is a column vector with the first element equal to one, and the rest equal to zero. Of course, there is a fundamental difference between computing β_0 for the $N = 0$ case, and using a high order estimator ($N > 0$) and then effectively discarding all $\boldsymbol{\beta}_n$'s except β_0. Unlike the former case, the high regression order method computes estimates of pixel values assuming an N-th order local polynomial structure is present by including higher order polynomial bases as in the matrix $\mathbf{X}$ (1.31). Similar to the 1-D case, for $N = 0$, the kernel estimator (1.33) is expressed as

$$\hat{z}(\mathbf{x}) = \hat{\beta}_0 = \frac{\sum_{i=1}^{P} K_{\mathbf{H}}(\mathbf{x}_i - \mathbf{x})\, y_i}{\sum_{i=1}^{P} K_{\mathbf{H}}(\mathbf{x}_i - \mathbf{x})}, \tag{1.34}$$

which is *Nadaraya-Watson* estimator (NWE) in 2-D. For example, when we choose the kernel function as Gaussian (1.26), NWE is nothing but the well-known Gaussian low-pass filter [29] and it provides a pixel value of interest β_0 by a weighted *linear* combination of the nearby samples. Even the higher order estimator can be generally expressed in the weight linear fashion as

$$\hat{z}(\mathbf{x}) = \hat{\beta}_0 = \mathbf{e}_1^T \widehat{\mathbf{b}} = \sum_{i=1}^{P} W_i(K, \mathbf{H}, N, \mathbf{x}_i - \mathbf{x})\, y_i, \tag{1.35}$$

where

$$\sum_{i=1}^{P} W_i(\cdot) = 1, \tag{1.36}$$

and we call W_i the *equivalent kernel* function for y_i (q.v. Appendix A and [3] for more detail). It is worth noting that the estimator (1.32) also yield local gradients for the regression orders $N > 0$ (q.v. for Appendix B). While the exact expressions in Appendices A and B yield the mathematical property of the kernel regression estimator, the pixel estimator and the gradient estimator can be simply expressed as follows. We can rewrite the overall estimator (1.32) for the regression order $N > 0$ as

$$\widehat{\mathbf{b}} = \left(\mathbf{X}^T \mathbf{K} \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{K}\, \mathbf{y} = \mathbf{W}_N\, \mathbf{y} = \begin{bmatrix} \mathbf{w}_0^T(N) \\ \mathbf{w}_1^T(N) \\ \mathbf{w}_2^T(N) \\ \vdots \end{bmatrix} \mathbf{y}, \tag{1.37}$$

where N is the regression order, and $\mathbf{w}_0(N)$, $\mathbf{w}_1(N)$, and $\mathbf{w}_2(N) \in \mathscr{R}^{P\times 1}$ are the *equivalent kernel* weight matrices that compute the unknown pixel value and its derivatives as follows. From (1.35) and (B.1), we have

$$\hat{z}(\mathbf{x}) = \hat{\beta}_0 = \mathbf{e}_1^T \widehat{\mathbf{b}} = \mathbf{w}_0^T(N)\,\mathbf{y}, \tag{1.38}$$

$$\nabla \hat{z}(\mathbf{x}) = \hat{\boldsymbol{\beta}}_1 = \begin{bmatrix} \mathbf{e}_2^T \\ \mathbf{e}_3^T \end{bmatrix} \widehat{\mathbf{b}} = \begin{bmatrix} \mathbf{w}_1^T(N) \\ \mathbf{w}_2^T(N) \end{bmatrix} \mathbf{y}. \tag{1.39}$$

Note that when we estimate the n-th derivatives of $z(\cdot)$, the regression order N must be equal or higher than n ($N \geq n$). For instance, $\mathbf{w}_1(0)$ and $\mathbf{w}_2(0)$ do not exist.

Therefore, regardless of the regression order, the classic kernel regression is local weighted averaging of data (linear filtering), where the order determines the type of complexity of the weighing scheme. This also suggests that higher order regressions ($N > 0$) are equivalents of the zeroth order regression ($N = 0$) but with a more complex equivalent kernel function. In other words, to effect the higher order regressions, the original kernel $K_{\mathbf{H}}(\mathbf{x}_i - \mathbf{x})$ is modified to yield a newly adapted "equivalent" kernel [3, 17, 30]

To have a better intuition of equivalent kernels, we study the example in Figure **1.8**, which illustrates a regularly sampled data set, and a horizontal cross section of its corresponding equivalent kernels for the regression orders $N = 0$, 1, and 2. The direct result of the symmetry condition (1.4) on K with uniformly sampled data is that all odd-order moments consist of element with values very close to zero (i.e., $\sum_i (\mathbf{x}_i - \mathbf{x}) K_{\mathbf{H}}(\mathbf{x}_i - \mathbf{x}) \approx 0$ in (A.9) and (A.10) in Appendix A). Therefore, as noted in Figure **1.8**(b), the equivalent kernels for $N = 0$ and $N = 1$ are essentially identical. As this observation holds for all regression orders, for *the regularly sampled data*, the $N = 2\nu$ (even number, i.e., $N = 0, 2, \cdots$) order regression is preferred to the computational more complex than the $N = 2\nu + 1$ (odd number, i.e., $N = 1, 3, \cdots$) order regression, as they produce the same equivalent kernels. This property manifests itself in Figure **1.8**, where

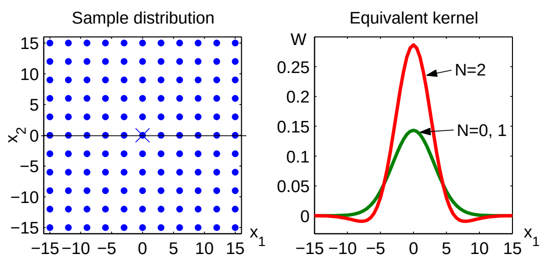


Figure 1.8: Equivalent kernels for a regularly sampled data: (a) a regularly sampled data set and (b) a horizontal cross section of the equivalent kernels of orders $N = 0$, 1, and 2. Gaussian kernel $K_\mathbf{H}$ in (1.26) is used with the smoothing matrix $\mathbf{H} = \text{diag}[10, 10]$.

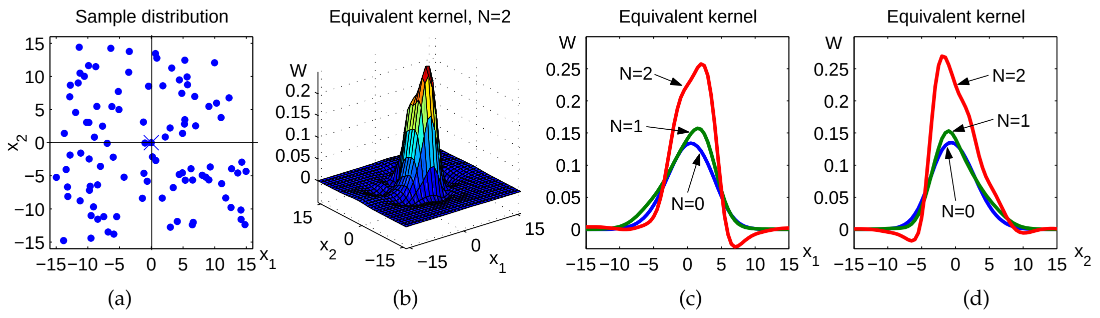


Figure 1.9: Equivalent kernels for an irregularly sampled data: (a) an irregularly sampled data set, (b) the second order equivalent kernel ($N = 2$), and (c)(d) the horizontal and vertical cross sections of the equivalent kernels of different orders ($N = 0, 1, 2$) are compared, respectively. Similar to the case of the regularly sampled data set in Figure **1.8**, Gaussian kernel $K_\mathbf{H}$ in (1.26) is also used with the smoothing matrix $\mathbf{H} = \text{diag}[10, 10]$.

the $N = 0$ or $N = 1$ ordered equivalent kernels are identical. It is also worth noting here that the side lobes of the second order equivalent kernel coefficients are negative even though we use Gaussian function. The negative coefficients in fact yield mild sharpening effect to the estimated images. With this property, we are able to locally control the amount of sharpening effect by the regression order. We study and take advantages of the property in Section 4.2.

In the next experiment, we compare the equivalent kernels for an irregularly sampled data set shown in Figure **1.9**(a). The second order equivalent kernel ($N = 2$) for

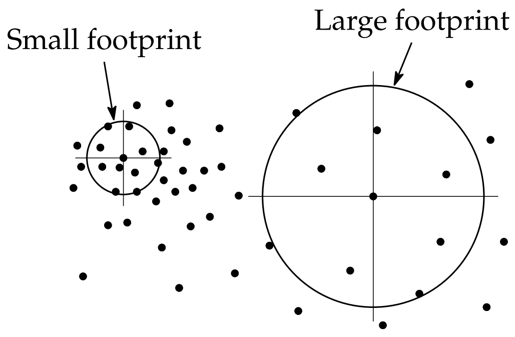


Figure 1.10: Smoothing (kernel support size) selection by local sample density.

the sampled marked with "×", is shown in Figure **1.9**(b). Figures **1.9**(c) and (d) show the horizontal and vertical cross sections of the equivalent kernels for the irregularly measured samples, respectively. This figure demonstrates the fact that the equivalent kernels tend to adapt themselves to the density of available samples. Also, unlike the uniformly sampled data case, since the odd-order moments are nonzero, the zeroth ($N = 0$) and first ($N = 1$) equivalent kernels are no longer identical.

1.3.4 Smoothing Matrix Selection

The shape (or contour) of the regression kernel as defined in (1.25), and consequently the performance of the estimator depends on the choice of the smoothing matrix $\mathbf{H}$ [16]. For the bivariate regression problem, the smoothing matrix $\mathbf{H}$ is 2×2, and it should extend the support of the regression kernel to contain "enough" samples. As illustrated in Figure **1.10**, it is reasonable to use a smaller support size in the area with more available samples, whereas a larger support size is more suitable for the more sparsely sampled area of the image.

The cross validation "leave-one-out" method [3, 13] is a popular technique for estimating the elements of the local smoothing matrices $\mathbf{H}_i$ for all the given samples y_i.

However, as the cross validation method is computationally very expensive, we can use a simplified and computationally more efficient model of the smoothing matrix as

$$\mathbf{H}_i = h\mu_i \mathbf{I}, \tag{1.40}$$

where μ_i is a scalar that captures the local density of samples (nominally set to $\mu_i = 1$) and h is the *global smoothing parameter*.

The global smoothing parameter is directly computed from the cross validation method, by minimizing the following cost function

$$C_{\text{cv}}(h) = \frac{1}{P}\sum_{i=1}^{P}\left\{\hat{z}_{i-}(\mathbf{x}_i) - y_i\right\}^2 \tag{1.41}$$

where $\hat{z}_{i-}(\mathbf{x}_i)$ is the estimated pixel values without including the i-th sample (i.e., y_i) at $\mathbf{x}_i$. To further reduce the computations, rather than leaving a single sample out, we can leave out a set of samples (a whole row or column) [31, 32, 33].

Following [11], the *local density parameter* μ_i is estimated as follows

$$\mu_i = \left\{\frac{\hat{f}(\mathbf{x}_i)}{\exp\left(\frac{1}{P}\sum_{i=1}^{P}\log\hat{f}(\mathbf{x}_i)\right)}\right\}^{-\zeta}, \tag{1.42}$$

where the sample density $\hat{f}(\mathbf{x}_i)$ is measured as

$$\hat{f}(\mathbf{x}_i) = \frac{1}{P}\sum_{i=1}^{P} K_{\mathbf{H}_i}(\mathbf{x}_i - \mathbf{x}), \tag{1.43}$$

and ζ, the *density sensitivity*, is a scalar satisfying $0 < \zeta \leq 1$. Note that $\mathbf{H}_i$ and μ_i are estimated in an iterative fashion. In the first iteration, μ_i is initialized by 1 and we estimate the density by (1.43). Then, we update μ_i by (1.42) with the estimated density and estimate the density. The process is repeated until μ_i converges (typically, only a few iterations (at most 5 iterations)).

Summary— In this chapter, we introduced a variety of image and video processing problems as regression and studied the classic kernel regression framework. We also showed that the framework is equivalent to an adaptive locally "linear" filtering process. The price that one pays for using such computationally efficient classic kernel regression methods with diagonal matrix $\mathbf{H}_i$ is the low quality of reconstruction in the edge areas. Experimental results on this material will be presented later in Chapter 2. Also in Chapter 2, we gain even better performance by proposing data-adaptive kernel regression methods which take into account not only the spatial sampling density of the data, but also the actual (pixel) values of those samples. These more sophisticated methods lead to locally adaptive "nonlinear" extensions of classic kernel regression.

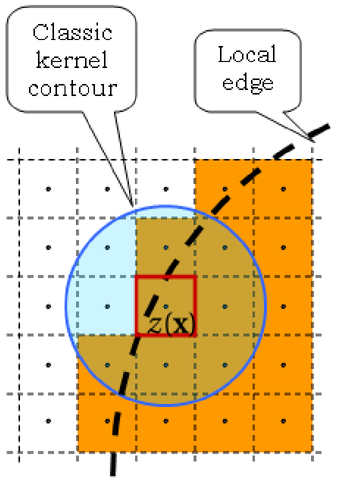
Chapter 2

Data-Adaptive Kernel Regression

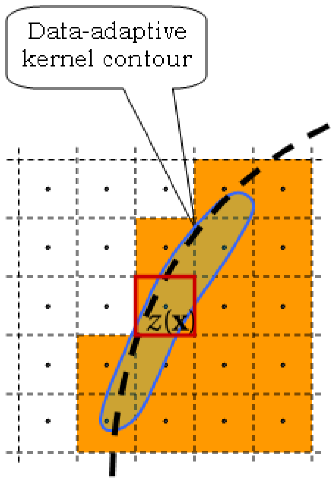
Abstract— In the previous chapter, we studied the classic kernel regression framework and its properties, and also showed its usefulness for image restoration and reconstruction purposes. One fundamental improvement on the above method can be realized by noting that, the local polynomial kernel regression estimates, independent of the regression order N, are always local "linear" combinations of the data. As such, though elegant, relativity easy to analyze, and with attractive asymptotic properties [16], they suffer from an inherent limitation due to this local linear action on the data. In what follows, we discuss extensions of the kernel regression method that enable this structure to have nonlinear, more effective, action on the given data: data-adaptive kernel regression.

2.1 Data-Adaptive Kernels

Data-adaptive kernel regression relies on not only the spatial properties (the sample location and density), but also the photometric properties of these samples (i.e.



(a) Contours of classic kernels



(b) Contours of data-adaptive kernels

Figure 2.1: Kernel contours in a uniformly sampled data set: (a) Kernels in the classic method depend only on the spatial distances, and (b) Data-adaptive kernels elongate with respect to the local edge structure.

pixel values). Thus, the effective size and shape of the regression kernel are adapted locally to image feature such as edges. A desired property of such regression kernel is illustrated in Figure **2.1**, in which the classic kernel estimates the pixel $z(\mathbf{x})$ by the combination of neighboring samples with linear weights while the data-adaptive kernel elongates/spreads along the local edge structure and the estimate is most strongly affected by the edge pixels. Hence, the data-adaptive kernel approach effectively suppresses noise while preserving local image structures.

Data-adaptive kernel regression is formulated similarly to (1.24) as an optimization problem

$$\min_{\{\boldsymbol{\beta}_n\}} \sum_{i=1}^{P} \left[y_i - \beta_0 - \boldsymbol{\beta}_1^T (\mathbf{x}_i - \mathbf{x}) - \boldsymbol{\beta}_2^T \text{vech}\left\{ (\mathbf{x}_i - \mathbf{x})(\mathbf{x}_i - \mathbf{x})^T \right\} - \cdots \right]^2 K_{\text{adapt}}(\mathbf{x}_i - \mathbf{x}, y_i - y), \quad (2.1)$$

where the data-adaptive kernel function K_{adapt} now depends on the spatial sample coordinates $\mathbf{x}_i$'s and density as well as the photometric values y_i of the data. Katkovnik *et* *al.* have also studied this data-adaptive approach in [34]. In the following, we study two different data-adaptive kernels: bilateral kernel and steering kernel, and discuss their properties.

2.1.1 Bilateral Kernel

A simple and intuitive choice of the K_{adapt} is to use separate terms for penalizing the spatial distance between the pixel position of interest $\mathbf{x}$ and its neighboring pixel positions $\{\mathbf{x}_i\}$, and the photometric "distance" between the pixel of interest y and its neighbors $\{y_i\}$:

$$K_{\text{bilat}}(\mathbf{x}_i - \mathbf{x}, y_i - y) \equiv K_{\mathbf{H}_{\text{s}}}(\mathbf{x}_i - \mathbf{x}) \cdot K_{h_{\text{p}}}(y_i - y), \tag{2.2}$$

where $\mathbf{H}_{\text{s}}$ $(= h_{\text{s}}\mathbf{I})$ is the spatial smoothing (diagonal) matrix and h_{p} is the photometric smoothing scalar. Figure **2.2** illustrates weight values for this bilateral kernel function at a few different regions of the clean Lena image: flat, edge, and Lena's eye. As seen in the figure, the photometric kernel captures local image structures effectively. The properties of this adaptive method, which we call *bilateral kernel regression* (for reasons that will become clear shortly), can be better understood by studying the special case of the zeroth order ($N = 0$), which results in a data-adapted version of the Nadaraya-Watson estimator (NWE):

$$\hat{z}(\mathbf{x}) = \hat{\beta}_0 = \frac{\sum_{i=1}^{P} K_{\mathbf{H}_{\text{s}}}(\mathbf{x}_i - \mathbf{x}) K_{h_{\text{p}}}(y_i - y)\, y_i}{\sum_{i=1}^{P} K_{\mathbf{H}_{\text{s}}}(\mathbf{x}_i - \mathbf{x}) K_{h_{\text{p}}}(y_i - y)}. \tag{2.3}$$

Interestingly, this is nothing but the well-studied and popular *bilateral filter* [7, 8]. We note that, in general, since the pixel values (y) at an arbitrary position ($\mathbf{x}$) might be unavailable from the given data, the direct application of the bilateral ker-

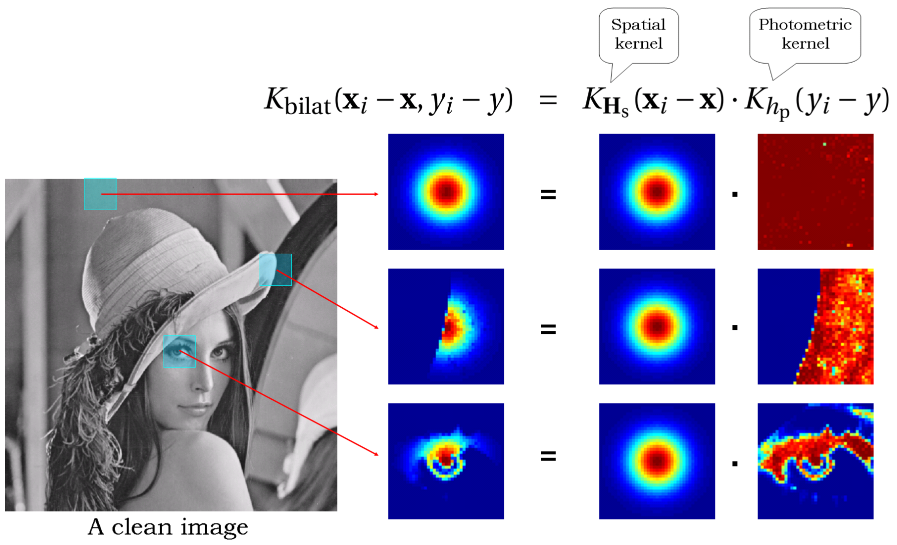

Spatial kernel
Photometric kernel
$K_{\text{bilat}}(\mathbf{x}_i - \mathbf{x}, y_i - y) = K_{\mathbf{H}_s}(\mathbf{x}_i - \mathbf{x}) \cdot K_{h_p}(y_i - y)$
A clean image

Figure 2.2: Bilateral kernel weights given by (2.2) at flat, edge, and Lena's eye regions of a clean image.

nel function (2.2) is limited to the denoising problem. This limitation, however, can be overcome by using an initial estimate of y by an appropriate interpolation technique. Also, it is worth noting that the bilateral kernel choice, along with higher order choices for N (> 0), will lead to generalizations of the bilateral filter, which we study in the following.

Similar to classic kernel regression, the pixel estimator given by bilateral kernel regression is also summarized as the form of the weighted linear combination of all the neighboring samples using the bilateral "equivalent" weight function W_i regardless of the regression order N as follows:

$$\hat{z}(\mathbf{x}) = \hat{\beta}_0 = \sum_{i=1}^{P} W_i(K, \mathbf{H}_s, h_p, N, \mathbf{x}_i - \mathbf{x}, y_i - y)\, y_i. \tag{2.4}$$

Figure **2.3** illustrates the bilateral equivalent weight function $W_i(K, \mathbf{H}_s, h_p, N, \mathbf{x}_i - \mathbf{x}, y_i - y)$ in (2.4) at a variety of image structures for the zeroth and second orders ($N = 0$ and 2). Note that each weigh function is respectively normalized. Figure **2.4** illustrates the

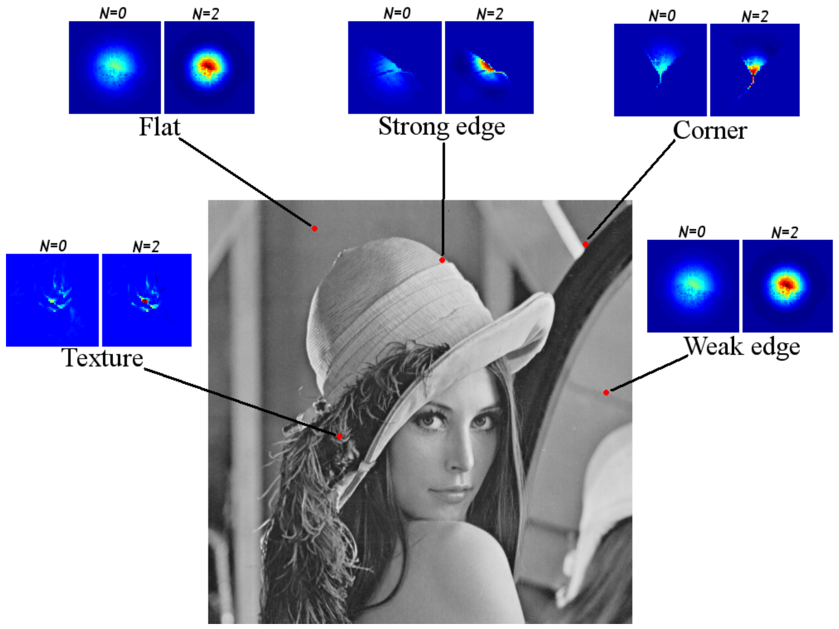
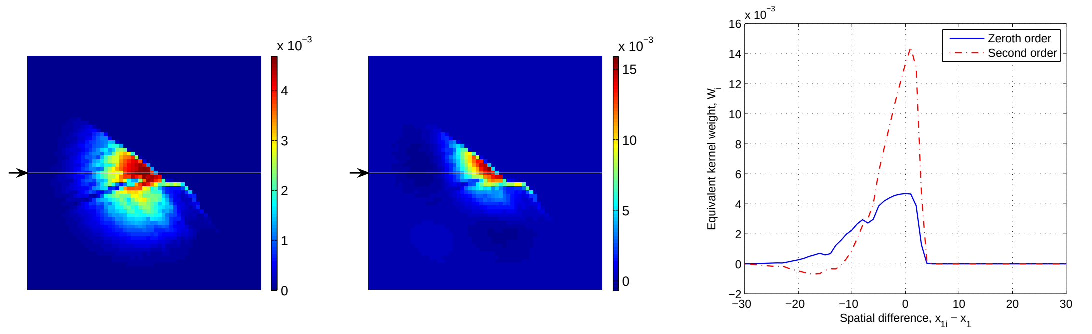


Figure 2.3: A visual analysis of the bilateral equivalent weight function $W_i(K, \mathbf{H}_s, h_p, N, \mathbf{x}_i - \mathbf{x}, y_i - y)$ in (2.4) at a variety of image structures; flat, strong edge, corner, texture, and weak edge for the zeroth and second order ($N = 0$ and 2). Note: Each weight function is respectively normalized, and Figure **2.4** illustrates the detail of the weight function at the strong edge.

details of W_i at the strong edge: (a)-(b) equivalent weight values W_i for the zeroth and second orders, respectively, and (c) the horizontal cross-sections as indicated in (a) and (b).

The derivation of the zeroth order bilateral filter (2.3) above indicates that the bilateral filter only consists of the constant term (i.e. β_0), and the consideration of only β_0 is the cause of filtered image signals being piecewise constant. Such signals are often unsuitable for image processing because image signals of interest have complicated contours, such as texture and gradation. Higher order bilateral filters relax the piecewise constancy by the choice of $N > 0$. For instance, the images estimated by the first and second order bilateral filter become locally linear and quadratic, respectively.



(a) Zeroth order (b) Second order (c) Horizontal cross-sections of W_i

Figure 2.4: Horizontal cross-sections of the bilateral equivalent weight function $W_i(K, \mathbf{H}_\mathrm{s}, h_\mathrm{p}, N, \mathbf{x}_i - \mathbf{x}, y_i - y)$ at the strong edge for the zeroth and second order ($N = 0$ and 2): (a)-(b) the footprints of W_i for the zeroth and second order, respectively, (c) the horizontal cross-sections pointed by the arrows of W_i of (a) and (b).

As a further extension of the standard bilateral filter, Elad suggested iterative filtering in order to intensify the smoothing effect in [8]. The iterative filtering process is as follows: (i) apply bilateral filter to the given noisy data, (ii) apply bilateral filter to the previous estimate, (iii) repeat the step (ii). For $N = 0$, such estimator can be written as

$$\hat{z}^{(\ell+1)}(\mathbf{x}) = \frac{\sum_{i=1}^{P} K_{\mathbf{H}_\mathrm{s}}(\mathbf{x}_i - \mathbf{x}) K_{h_\mathrm{p}}\left(\hat{z}^{(\ell)}(\mathbf{x}_i) - \hat{z}^{(\ell)}(\mathbf{x})\right) \hat{z}^{(\ell)}(\mathbf{x}_i)}{\sum_{i=1}^{P} K_{\mathbf{H}_\mathrm{s}}(\mathbf{x}_i - \mathbf{x}) K_{h_\mathrm{p}}\left(\hat{z}^{(\ell)}(\mathbf{x}_i) - \hat{z}^{(\ell)}(\mathbf{x})\right)}, \tag{2.5}$$

where $\hat{z}^{(0)}(\mathbf{x}_i) = y_i$ and ℓ is the index of the number of iterations. This filtering algorithm is very similar to Mean-Shift algorithm [35, 36], in which the spatial kernel function $K_{\mathbf{H}_\mathrm{s}}(\mathbf{x}_i - \mathbf{x})$ is not taken into account.

The bilateral filter has appeared in another form (2.3), which is known as the *Susan filter* [37]. The difference between bilateral filter and Susan filter is minor; Susan filter excludes the center pixel from the estimates. That is to say, Susan filter is expressed as

$$\hat{z}(\mathbf{x}) = \hat{\beta}_0 = \frac{\sum_{\mathbf{x}_i \neq \mathbf{x}} K_{\mathbf{H}_\mathrm{s}}(\mathbf{x}_i - \mathbf{x}) K_{h_\mathrm{p}}(y_i - y)\, y_i}{\sum_{\mathbf{x}_i \neq \mathbf{x}} K_{\mathbf{H}_\mathrm{s}}(\mathbf{x}_i - \mathbf{x}) K_{h_\mathrm{p}}(y_i - y)}. \tag{2.6}$$

This small modification significantly improves the filter performance in particular when the given data contains a few outliers (e.g. salt & pepper noise). For the bilateral filter, such outlier pixels yields very small photometric kernel values for neighboring pixels because the photometric distances, $y_i - y$, tend to be large. In other words, the bilateral filter doesn't smooth an outlier pixel with its neighboring pixels. We will discuss an alternative approach for removing outliers in Section 2.3 including comparisons to Susan filter.

In any event, breaking K_{adapt} into spatial and photometric terms as utilized in the bilateral case weakens the estimator performance since it limits the degrees of freedom and ignores correlations between positions of the pixels and their values. In particular, we note that, for very noisy data sets, the photometric distances, $y_i - y$, tend to be large and noisier. Therefore, most photometric weights are close to zero and also noisy as shown in Figure **2.5**. Such weights are effectively useless. Although we could set the photometric smoothing parameter (h_p) larger in order to reduce the effect of the noisy photometric distances, the bilateral filter becomes almost equivalent to the non-linear (Gaussian low-pass) filter with a large h_p. The following section provides a general solution to overcome this and many other drawbacks of competing approaches.

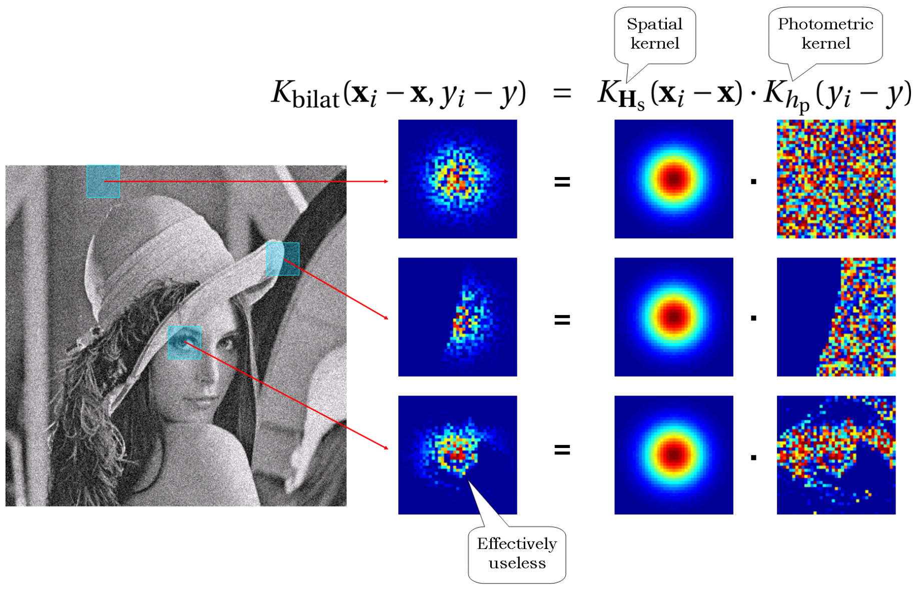

Spatial kernel
Photometric kernel
$K_{\text{bilat}}(\mathbf{x}_i - \mathbf{x}, y_i - y) = K_{\mathbf{H}_s}(\mathbf{x}_i - \mathbf{x}) \cdot K_{h_p}(y_i - y)$
Effectively useless

Figure 2.5: Bilateral kernel weights given by (2.2) at flat, edge, and Lena's eye regions of a noisy image. The noisy image is given by adding white Gaussian noise with standard deviation = 25 (the corresponding SNR is 5.64[dB]).

2.1.2 Steering Kernel

The filtering procedure that we propose next takes the data-adaptive idea one step further, based upon the earlier nonparametric framework. In particular, we observe that the effect of computing the photometric kernel, $K_{h_p}(y_i - y)$ in (2.2) is to implicitly measure a function of the local gradient estimated between neighboring pixels and to use this estimate to weight the respective measurements. As an example, if a pixel is located near an edge, then pixels on the same side of the edge will have much stronger influence in the filtering. With this an initial estimate of the image gradients is made using some kind of gradient estimator (say the second order classic kernel regression method). Next, this estimate is used to measure the dominant orientation of the local gradients in the image (e.g. [38]). In a second filtering stage, this orientation information is then used to adaptively "steer" the local kernel, resulting in elongated, contours spread along the directions of the local edge structure. With these locally adapted kernels, the denoising is effected most strongly along the edges, rather than across them, resulting in strong preservation of details in the final output. To be more specific, the data-adaptive kernel function takes the form

$$K_{\text{steer}}(\mathbf{x}_i - \mathbf{x}, y_i - y) \equiv K_{\mathbf{H}_i^{\text{steer}}}(\mathbf{x}_i - \mathbf{x}), \tag{2.7}$$

where $\mathbf{H}_i^{\text{steer}}$'s are now the data-dependent full (2×2) matrices which we call *steering* matrices. We define them as

$$\mathbf{H}_i^{\text{steer}} = h\mu_i \mathbf{C}_i^{-\frac{1}{2}}, \tag{2.8}$$

where again h and μ_i are the global smoothing parameter and the local density parameter, respectively, and $\mathbf{C}_i$'s are (symmetric, 2×2) covariance matrices based on differences in the local gray-values. A good choice for $\mathbf{C}_i$ will effectively spread the kernel function along the local edges, as shown in Figure **2.1**(b). It is worth noting that, even if we choose a large h in order to have a strong denoising effect, the undesirable blurring effect, which would otherwise have resulted, is tempered around edges with appropriate choice of $\mathbf{C}_i$. With such steering matrices, for example, if we choose a Gaussian kernel, i.e. plugging the steering matrix (2.8) into Gaussian kernel function (1.26), the steering kernel is mathematically represented as

$$K_{\mathbf{H}_i^{\text{steer}}}(\mathbf{x}_i - \mathbf{x}) = \frac{\sqrt{\det(\mathbf{C}_i)}}{2\pi h^2 \mu_i^2} \exp\left\{ -\frac{(\mathbf{x}_i - \mathbf{x})^T \mathbf{C}_i (\mathbf{x}_i - \mathbf{x})}{2h^2\mu_i^2} \right\}. \tag{2.9}$$

It is also noteworthy that, for the estimate of the unknown pixel β_0 $(= z(\mathbf{x}))$, the steering kernel function takes *all* the steering matrices ($\mathbf{H}_i^{\text{steer}}$) of the neighboring pixels y_i around the position of interest $\mathbf{x}$ into account, and hence, the steering kernel is not simply elliptic but it provides us weights that fit the local image structures more flexibly. We will show some actual steering kernels shortly in this section.

The local edge structure is related to the gradient covariance (or equivalently, the locally dominant orientation), where a naive estimate of this covariance matrix may be obtained as follows:

$$\mathbf{C}_i^{\text{naive}} = \mathbf{J}_i^T \mathbf{J}_i, \tag{2.10}$$

where $\mathbf{J}_i$ is a stack of local gradient vectors in a local analysis window ω_i:

$$\mathbf{J}_i = \begin{bmatrix} \vdots & \vdots \\ z_{x_1}(\mathbf{x}_j) & z_{x_2}(\mathbf{x}_j) \\ \vdots & \vdots \end{bmatrix}, \quad \mathbf{x}_j \in \omega_i, \tag{2.11}$$

$z_{x_1}(\cdot)$ and $z_{x_2}(\cdot)$ are the first derivatives along x_1 (vertical) and x_2 (horizontal) directions, and ω_i is a local analysis window around the position of a given sample. The dominant local orientation of the gradients is then related to the eigenvectors of this estimated matrix. Since the gradients, $z_{x_1}(\cdot)$ and $z_{x_2}(\cdot)$, depend on the pixel values $\{y_i\}$, and since the choice of the localized kernels in turns depends on these gradients, it, therefore, follows that the "equivalent" kernels for the proposed data-adaptive methods form a locally "nonlinear" combination of the data:

$$\hat{z}(\mathbf{x}) = \sum_{i=1}^{P} W_i(K, \mathbf{H}_i^{\text{steer}}, N, \mathbf{x}_i - \mathbf{x})\, y_i. \tag{2.12}$$

While the above approach to computing the steering matrices [38, 39, 40], is simple and has nice tolerance to noise, the resulting estimate can be unstable, and, therefore, care must be taken not to take the inverse of the estimate directly in this case. In such case, a diagonal loading or regularization methods can be used to obtain stable estimates of the covariance. In [38], Feng *et al.* proposed an effective *multiscale* technique for estimating local orientations, which fits the requirements of this problem nicely. Informed by the above, in this work, we take a more robust approach to the design of the steering matrix.

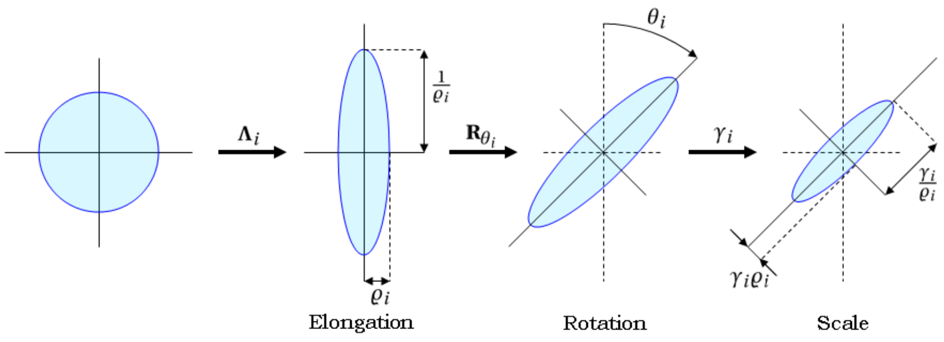


Figure 2.6: A schematic representation illustrating the effects of the steering matrix and its components ($\mathbf{C}_i = \gamma_i \mathbf{R}_{\theta_i} \mathbf{\Lambda}_i \mathbf{R}_{\theta_i}^T$) on the size and shape of the regression kernel footprint.

In order to have a more convenient form of the covariance matrix, we decompose it into three components (equivalent to eigenvalue decomposition) as follows:

$$\mathbf{C}_i = \gamma_i \mathbf{R}_{\theta_i} \mathbf{\Lambda}_i \mathbf{R}_{\theta_i}^T, \tag{2.13}$$

where $\mathbf{R}_{\theta_i}$ is the rotation matrix and $\mathbf{\Lambda}_i$ is the elongation matrix:

$$\mathbf{R}_{\theta_i} = \begin{bmatrix} \cos\theta_i & \sin\theta_i \\ -\sin\theta_i & \cos\theta_i \end{bmatrix}, \quad \mathbf{\Lambda}_i = \begin{bmatrix} \varrho_i & 0 \\ 0 & \frac{1}{\varrho_i} \end{bmatrix}. \tag{2.14}$$

Now, the covariance matrix given by the three parameters γ_i, θ_i, and ϱ_i, which are the scaling, rotation, and elongation parameters, respectively. Figure **2.6** schematically explains how these parameters affect the spreading of kernels. First, the circular kernel is elongated by the elongation matrix $\mathbf{\Lambda}_i$, and its semi-minor and major axes are given by ϱ_i. Second, the elongated kernel is rotated by the matrix $\mathbf{R}_{\theta_i}$. Finally, the kernel is scaled by the scaling parameter γ_i.

We define the scaling, elongation, and rotation parameters as follow. Following the work in [38], the dominant orientation of the local gradient field is the singular vector corresponding to the smallest (nonzero) singular value of the local gradient matrix $\mathbf{J}_i$ (2.11) arranged in the following form:

$$\mathbf{J}_i = \mathbf{U}_i \mathbf{S}_i \mathbf{V}_i^T = \mathbf{U}_i \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix}^T, \tag{2.15}$$

where $\mathbf{U}_i \mathbf{S}_i \mathbf{V}_i^T$ is the truncated singular value decomposition of $\mathbf{J}_i$, and $\mathbf{S}_i$ is a diagonal 2×2 matrix representing the energy in the dominant directions. Then, the second column of the 2×2 orthogonal matrix $\mathbf{V}_i$, $\mathbf{v}_2 = [v_{12}, v_{22}]^T$, defines the dominant orientation angle θ_i as

$$\theta_i = \arctan\left(\frac{v_{12}}{v_{22}}\right). \tag{2.16}$$

That is, the singular vector corresponding to the smallest nonzero singular value s_2 of $\mathbf{J}_i$ represents the dominant orientation of the local gradient field. The elongation parameter ϱ_i can be scaled corresponding to the energy of the dominant gradient direction

$$\varrho_i = \frac{s_1 + \lambda'}{s_2 + \lambda'}, \quad \lambda' \geq 0, \tag{2.17}$$

where λ' is a "regularization" parameter for the kernel elongation, which dampens the effect of the noise and restricts the ratio from becoming degenerate. The intuition behind (2.17) is to keep the shape of the kernel circular in flat areas ($s_1 \approx s_2 \approx 0$), and elongate it near edge areas ($s_1 \gg s_2$). Finally, the scaling parameter γ_i is defined by

$$\gamma_i = \left(\frac{s_1 s_2 + \lambda''}{M}\right)^{\alpha}, \tag{2.18}$$

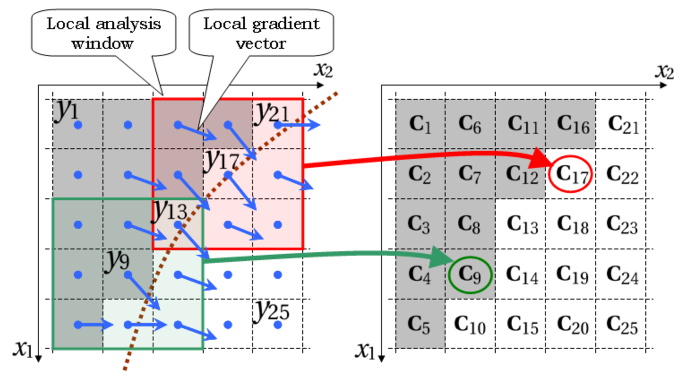
where λ'' is again a "regularization" parameter, which dampens the effect of the noise and keeps γ_i from becoming zero[1], M is the number of samples in the local analysis window, and α is the structure sensitivity parameter. The intuition behind (2.18) is that,

[1]The regularization parameters λ' and λ'' are used to prohibit the shape of the kernel from becoming infinitely narrow and long. In practice, it suffices to keep these numbers reasonably small, and, therefore, in all experiments in this chapter, we fixed their values equal to $\lambda' = 1.0$ and $\lambda'' = 0.01$, respectively.

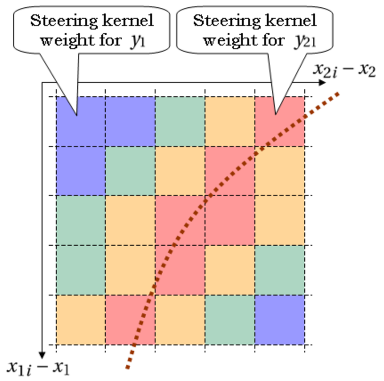
to reduce noise effects while producing sharp images, large footprints are preferred in the flat (smooth regions) and smaller ones in the textured areas. Note that the local gradients and the eigenvalues of the local gradient matrix $\mathbf{C}_i^{\text{naive}}$ are smaller in the flat (low-frequency) areas than the textured (high-frequency) areas. As the product $s_1 s_2$ is the geometric mean of the eigenvalues of $\mathbf{C}_i^{\text{naive}}$, γ_i makes the steering kernel area large in the flat, and small in the textured areas. The structure sensitivity α (typically $0 \leq \alpha \leq 0.5$) controls how strongly the size of the kernel footprints is affected by the local structure. The product of the singular values indicates the amount of energy of the local signal structure: the larger the product, the stronger and the more complex the local structure is. A large α is preferable when the given signal carries severe noise.

While the choice of the parameters in the above may appear to be ad-hoc, we direct the interested readers to a more careful statistical analysis of the distributional properties of the singular values (s_1, s_2) in [38, 41, 42]. Our particular selections for these parameters are directly motivated by these earlier works. However, to maintain focus, we have elected not to include such details in this presentation. We also note that the presented formulation is quite close to the apparently independently derived normalized convolution formulation of [6]. The significant difference between the adaptive normalized convolution and the steering kernel method is that the contours of the kernel function in adaptive normalized convolution method [6] is always elliptic.

Figure **2.7** illustrates a schematic representation of the estimate of local covariance matrices and the computation of steering kernel weights for the center pixel y_{13}. First, we estimate the gradients and compute the local covariance matrix $\mathbf{C}_i$ by (2.13)-(2.18) for each pixel. Then, when denoising y_{13}, we compute the steering kernel


Local analysis window
Local gradient vector
y_1
y_{21}
y_{17}
y_{13}
y_9
y_{25}
x_2
x_1
$\mathbf{C}_1$ $\mathbf{C}_6$ $\mathbf{C}_{11}$ $\mathbf{C}_{16}$ $\mathbf{C}_{21}$
$\mathbf{C}_2$ $\mathbf{C}_7$ $\mathbf{C}_{12}$ $\mathbf{C}_{17}$ $\mathbf{C}_{22}$
$\mathbf{C}_3$ $\mathbf{C}_8$ $\mathbf{C}_{13}$ $\mathbf{C}_{18}$ $\mathbf{C}_{23}$
$\mathbf{C}_4$ $\mathbf{C}_9$ $\mathbf{C}_{14}$ $\mathbf{C}_{19}$ $\mathbf{C}_{24}$
$\mathbf{C}_5$ $\mathbf{C}_{10}$ $\mathbf{C}_{15}$ $\mathbf{C}_{20}$ $\mathbf{C}_{25}$

(a) Covariance matrices from local gradients with 3×3 analysis window


Steering kernel weight for y_1
Steering kernel weight for y_{21}
$x_{2i} - x_2$
$x_{1i} - x_1$

(b) Steering kernel weights

Figure 2.7: A schematic representation of the estimates of local covariance metrics and the steering kernel weights at a local region with one dominant orientation: (a) First, we estimate the gradients and compute the local covariance matrix $\mathbf{C}_i$ by (2.13) for each pixel, and (b) Next, when denoising y_{13}, we compute the steering kernel weights with $\mathbf{C}_i$ for neighboring pixels. Even though, in this case, the spatial distances between y_{13} and y_1 and between y_{13} y_{21} are equal, the steering kernel weight for y_{21} (i.e. $K_{\mathbf{H}_{21}}(\mathbf{x}_{21} - \mathbf{x}_{13})$) is larger than the one for y_1 (i.e. $K_{\mathbf{H}_1}(\mathbf{x}_1 - \mathbf{x}_{13})$). This is because y_{13} and y_{21} are located along the same edge.

weights for each neighboring pixel with its $\mathbf{C}_i$. In this case, even though the spatial distances from y_{13} to y_1 and to y_{21} are equal, the steering kernel weight for y_{21} (i.e. $K_{\mathbf{H}_{21}}(\mathbf{x}_{21} - \mathbf{x}_{13})$) is larger than the one for y_i (i.e. $K_{\mathbf{H}_1}(\mathbf{x}_1 - \mathbf{x}_{13})$). Moreover, as Figure **2.8**(a) illustrates the steering kernel weights on a variety of image structures of a clean Lena image, weights given by the steering kernel function (2.9) with (2.13) captures local structures more effectively. This is because the steering kernel function is a function of the position of neighboring samples ($\mathbf{x}_i$) with the position of interest ($\mathbf{x}$) held fixed. Each neighboring sample (y_i) has a steering matrix ($\mathbf{H}_i^{\text{steer}}$), and, unlike the adaptive normalized convolution method, the steering kernel function takes not only the steering matrix at the position of interest but also its neighborhoods' into account. As a result, the steering kernel has more flexibility to adapt to local image structures. This property is effective for not only denoising and interpolation of images but also object

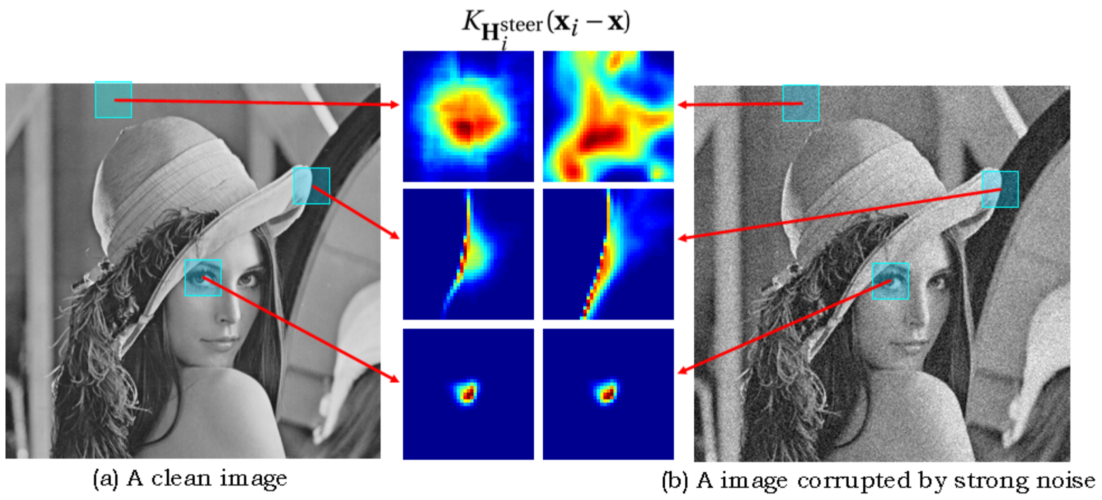

$K_{\mathbf{H}_i^{\text{steer}}}(\mathbf{x}_i - \mathbf{x})$
(a) A clean image
(b) A image corrupted by strong noise

Figure 2.8: Visualizations of steering kernels with covariance matrices $\mathbf{C}_i$ given by the local orientation estimate (2.13) at a variety of image structures. (a) the estimated kernel values in a clean image and (b) the estimated kernel values for the same areas of a noisy image (after 7 iterations considering additive Gaussian noise with standard deviation = 25 similar to the experiment in Section 2.2.

recognition applications [43]. In addition, even in a noisy case (a noisy Lena image given by adding white Gaussian noise with standard deviation = 25) illustrated in Figure **2.8**(b), the shape and orientation of of the kernels are very close to those of the noiseless case. Also, depending on the underlying features, in the flat areas, they are relatively more spread to reduce the noise effects, whereas, in the edge areas, they are spread along the local edge to reduce the noise effects along the edge rather and across them as described in Figure **2.9**.

Up to this point, we mostly described how the above formulation works for the denoising case. Figure **2.10** illustrates a summary of image upscaling by steering kernel regression. We begin with computing steering (covariance) matrices, $\mathbf{C}_i$ for all the pixels, y_i, from the input image shown in Figure **2.10**(a) by (2.13). Once $\mathbf{C}_i$'s are available, we have the steering matrices, $\mathbf{H}_i^{\text{steer}}$ by (2.8), and compute steering kernel

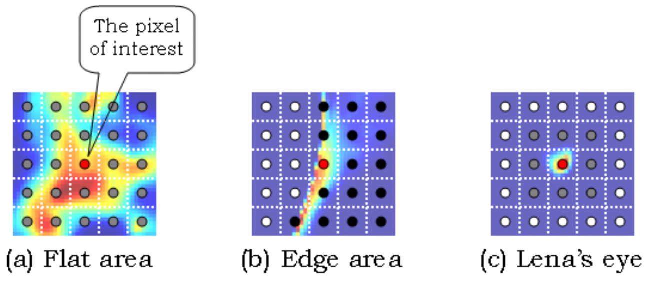
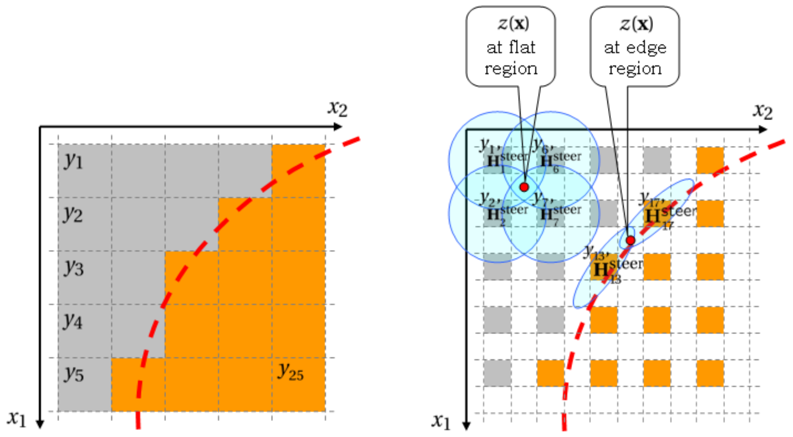


Figure 2.9: A schematic description of pixel denoising by steering kernel regression: (a) at the flat area, the steering kernel spreads widely, and steering kernel regression denoises the pixel of interest by a weighted average of neighboring samples (2.12). (b) For the edge area, the kernel spreads along the local edge, and the estimator computes an average of pixels on the same edge, which the pixel of interest is located on. (c) At a area with complex structures, e.g. Lena's eye, the steering kernel stays small. Thus the estimator effectively leave the pixel alone and preserve the local structure. The kernel weights are the ones shown in Figure **2.8**(b).

weights by (2.9). For example, when we estimate the missing pixel $z(\mathbf{x})$ at $\mathbf{x}$ shown as the red circle in Figure **2.10**(b), the steering kernel function gives high weights to the samples y_{13} and y_{17}. This is because $z(\mathbf{x})$ most likely lies on the same edge (shown by the red dashed curve) as y_{13} and y_{17}. Next, plugging the steering kernel weights into (1.33), we compute the equivalent kernel W_i and the estimator (2.12) gives the estimated pixels $\hat{z}(\mathbf{x})$ at $\mathbf{x}$. Furthermore, Figure **2.11** illustrates a schematic description of image reconstruction from irregularly sampled data using steering kernel regression. Figure **2.11**(a) shows the input data which are irregularly sampled. First, we estimate missing pixels on regular grid positions (marked as black dots) by classic kernel regression[2], and that is what we call "pilot estimation". Similar to the regular case explained in Figure **2.10**, we compute steering matrices for the given irregular data by (2.8) and (2.13). One simple way to estimate a steering matrix for an off-grid sample is to take the local analysis window at the nearest regular grid position. Once the steer-

[2]We use the second order classic kernel regression, but other simple methods can be of course used.


x_2
x_1
y_1
y_2
y_3
y_4
y_5
y_{25}
$z(\mathbf{x})$ at flat region
$z(\mathbf{x})$ at edge region
x_2
x_1
y_1, $\mathbf{H}_1^{\text{steer}}$
y_6, $\mathbf{H}_6^{\text{steer}}$
y_2, $\mathbf{H}_2^{\text{steer}}$
y_7, $\mathbf{H}_7^{\text{steer}}$
y_{17}, $\mathbf{H}_{17}^{\text{steer}}$
y_{13}, $\mathbf{H}_{13}^{\text{steer}}$

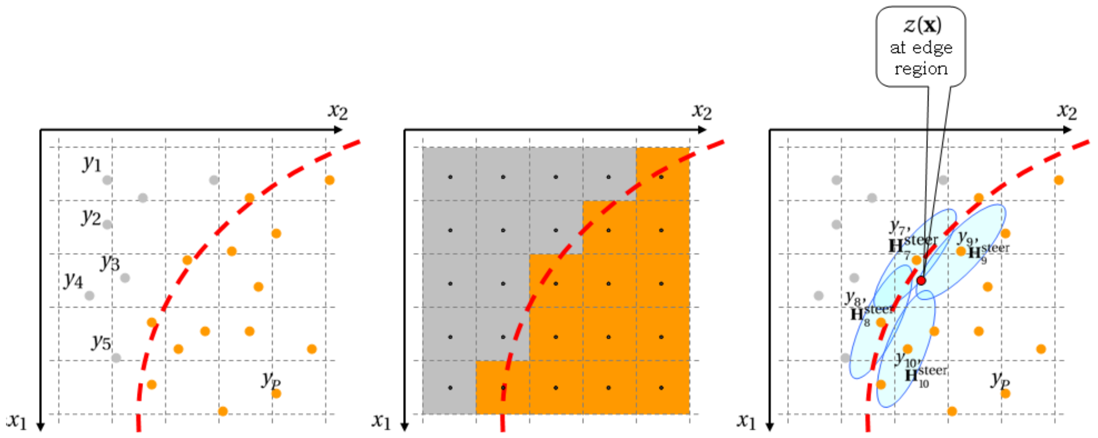
(a) Input image (b) The given samples with steering matrices

Figure 2.10: Steering kernel regression for image upscaling: (a) Input image, (b) we compute steering matrices for each given pixel and then estimate the missing pixel $z(\mathbf{x})$ and denoise the given pixels y_i. In the figure, we show regression examples at two positions: one is in flat region and the other is in edge region. When we estimate $z(\mathbf{x})$ in flat region, the steering kernels of neighboring pixels tend to be circular and widely spread, and thus the estimated pixel is the average value of the neighboring pixels (i.e. y_1, y_2, y_6, and y_7). On the other hand, in edge region, the steering kernels are elliptic and elongated along the local orientation, and the steering kernel regression estimates the missing pixel $z(\mathbf{x})$ by taking its neighboring pixels that are located on the same orientation (i.e. y_{13} and y_{17}) strongly into account. The red dashed line is a speculative local orientation.

ing matrices are available, we estimate the missing pixels of interest by steering kernel regression (2.12). Figure **2.11**(c) shows an example of estimating a pixel near an local edge in which steering kernel regression gives large weights to the neighboring samples located near the same local edge (i.e. y_7, y_8, y_9, and y_{10}). Hence, the steering kernel approach preserves and reconstructs the edge well.

2.1.3 Iterative Steering Kernel Regression

The estimated smoothing matrices of the steering kernel regression method are data dependent, and, consequently, sensitive to the noise in the input image. As



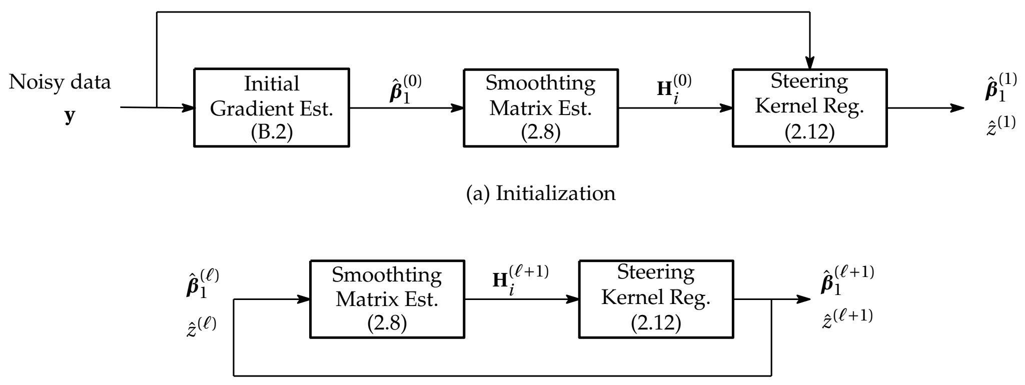
x_2
x_1
y_1
y_2
y_3
y_4
y_5
y_P
$z(\mathbf{x})$ at edge region
$y_7, \mathbf{H}_7^{steer}$
$y_8, \mathbf{H}_8^{steer}$
$y_9, \mathbf{H}_9^{steer}$
$y_{10}, \mathbf{H}_{10}^{steer}$

(a) Irregularly sampled data (b) Pilot estimation (c) Reconstruction by SKR

Figure 2.11: Steering kernel regression for image reconstruction from irregularly sampled data: (a) A irregularly sampled data set. (b) First we estimate missing pixels on regular grid positions (marked as black dots) by, for example, classic kernel regression, and that is what we call "pilot estimation". Then we estimate steering matrices for the given irregular data. One simple way to estimate a steering matrix for an off-grid sample is to take the local analysis window at the nearest regular grid position. (c) Once the steering matrices are available, we estimate the missing pixel of interest by steering kernel regression. The figure illustrate an example of estimating a pixel at near an local edge. This case, steering kernels give larger weights to the samples located on the same edge (i.e. y_7, y_8, y_9, and y_{10}).

we experimentally demonstrate in this section, steering kernel regression is most effective when an iterative regression/denosing procedure is used to exploit the output (less noisy) image of each iteration to estimate the photometric terms of the kernel in the next iteration. The most simple block diagram representation of this method motivated by the iterative bilateral filter method (2.5) is shown in Figure **2.12**, where ℓ is the iteration number. In the first diagram shown in Figure **2.12**(a), the data samples are used to create the initial (dense) estimte[3] of the interpolated output image. In the next iteration, the reconstructed (less noisy) image is used to calculate a more reliable

[3]Note that, in this thesis, all adaptive kernel regression experiments are initialized with the outcome of the second order classic kernel regression.

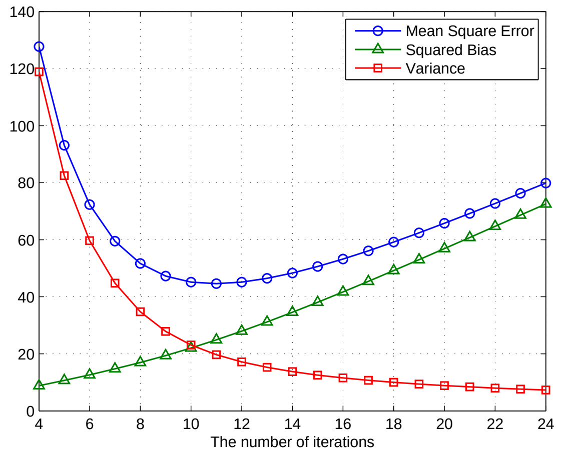

Noisy data
$\mathbf{y}$
Initial Gradient Est. (B.2)
$\hat{\boldsymbol{\beta}}_1^{(0)}$
Smoothting Matrix Est. (2.8)
$\mathbf{H}_i^{(0)}$
Steering Kernel Reg. (2.12)
$\hat{\boldsymbol{\beta}}_1^{(1)}$
$\hat{z}^{(1)}$
$\hat{\boldsymbol{\beta}}_1^{(\ell)}$
$\hat{z}^{(\ell)}$
Smoothting Matrix Est. (2.8)
$\mathbf{H}_i^{(\ell+1)}$
Steering Kernel Reg. (2.12)
$\hat{\boldsymbol{\beta}}_1^{(\ell+1)}$
$\hat{z}^{(\ell+1)}$

(a) Initialization

(b) Iteration

Figure 2.12: Block diagram representation of iterative steering kernel regression: (a) the initialization process, and (b) the iteration process.

estimate of the gradient as shown in Figure **2.12**(b), and this process continues for a few more iterations. A quick consultation with Figure **2.12**(a) shows that, although the iterative algorithm relies on an initial estimate of the gradient, we directly apply the estimated kernels on the original (noninterpolated) samples which results in the populated (or denoised) image in the first iteration. Therefore, denoising and interpolation can be done jointly in one step. Further iterations in Figure **2.12**(b) apply the modified kernels on the denoised pixels which results in more aggressive noise removal. In other words, the output image becomes more blurry as iterations continue. We note that while increasing the number of iterations reduces the variance of the estimate, it also leads to increased bias (which manifests as blurriness). Therefore, in some iterations, a minimum mean-squared estimate is obtained. An example of the typical behavior of mean square error (MSE), variance, and bias, and estimated images at the different number of iterations are shown in Figure **2.13**. Although, in the first several iterations, the MSE value decreases, as does the variance, the bias keeps increasing. At a certain number of iterations, the MSE hits bottom and increases as well as the



(a) MSE, bias, and variance of the iterative steering kernel regression method

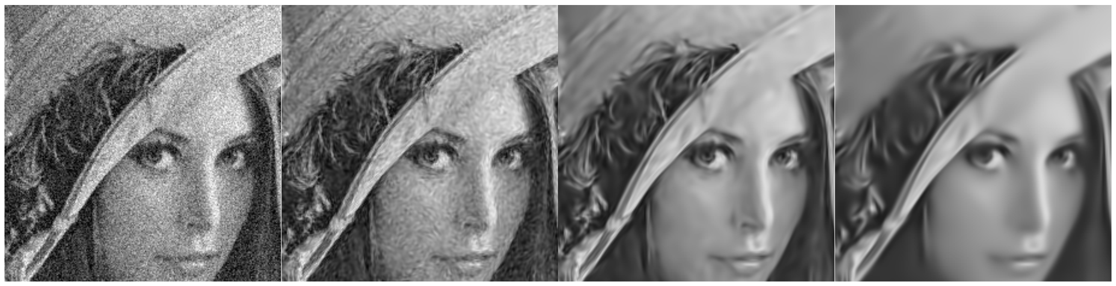

(b) Input image (MSE = 25^2) (c) 4 iterations (MSE = 126) (d) 11 iterations (MSE = 44) (e) 24 iterations (MSE = 80)

Figure 2.13: An example of the behavior of mean square error, variance, and bias of the iterative steering kernel method, and its estimated images at the different number of iterations.

bias. Another property of the iteration method is that, with a larger global smoothing parameter h in (2.8), we need fewer iterations to reach the minimum MSE. Another iteration scheme is to compute the steering matrices from the cleaned image, but to apply the SKR to the given noise-ridden image. In Section 6.2.3, we discuss this issue including a way to optimize the number of iterations as a future work.

2.2 Experiments

Having introduced the iterative steering kernel regression (SKR) above, in this section, we provide experiments on simulated and real data. These experiments show diverse applications, including denoising, deblocking, image reconstruction from irregularly sampled data, and resolution enhancement, illustrating the excellence of the proposed adaptive technique.

2.2.1 Image Restoration

In the first set of experiments, we compare the performance of several denoising techniques. We set up a controlled simulated experiment by adding white Gaussian noise with standard deviation 25 to the Lena image (512×512) shown in Figure. **2.14**(a), while the resulting noise-ridden image is shown in Figure. **2.14**(b) (the corresponding SNR value[4] is 5.64[dB]). The noise-ridden image is then denoised by the classic KR[5] (1.35) with $N = 2$ (second order) and $h = 1.8$, result of which is shown in Figure **2.14**(c). The result of applying the bilateral filter (2.3) with $h_s = 1.5$ and $h_p = 7.4$ is shown in Figure **2.14**(d). For the sake of comparison, we have included the result of applying anisotropic diffusion[6] [25], a wavelet denoising method[7] [46], K-SVD[8] [44], and BM3D[9] [45] in Figures **2.14**(e)-(h), respectively. Finally, Figure **2.14**(i) shows the result

[4]Signal-to-Noise Ratio is defined as $20\log_{10}(\sigma_{\text{signal}}/\sigma_{\text{noise}})$, where σ_{noise} and σ_{signal} are standard deviation of noise and clean signal, respectively.

[5]The criterion for parameter selection in this example (and other simulated examples discussed in this section) was to choose parameters which give the best mean square error result and the suggested parameter by the authors for K-SVD [44] and BM3D [45].

[6]The software is available at
`http://www.cns.nyu.edu/heegerlab/index.php?page=software&id=robanistdif`.

[7]The software is available at `http://decsai.ugr.es/~javier/denoise/index.html`.

[8]The software is available at `http://www.cs.technion.ac.il/~elad/software/`.

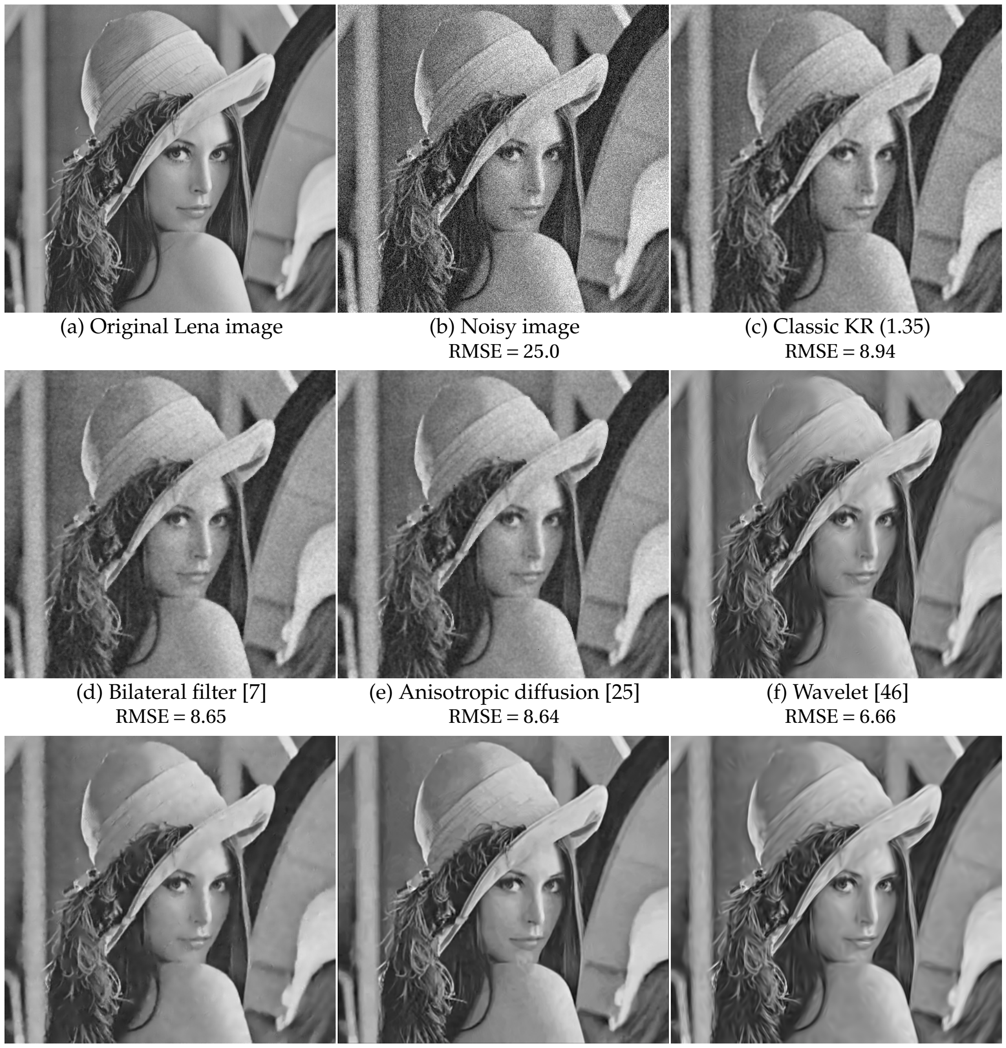
[9]The software is available at `http://www.cs.tut.fi/~foi/GCF-BM3D/`.

of applying the iterative SKR proposed in Figure **2.12** with $N = 2$, $h = 2.5$, and 7 iterations. The corresponding RMSE[10] values of the restored images of Figures **2.14**(c)-(i) are (c)8.94, (d)8.65, (e)8.64, (f)6.66, (g)6.90, (h)6.35, and (i) 6.64, respectively. Although, the iterative SKR is not specifically designed to remove white Gaussian noise, the result is comparable to the state of art denoising methods numerically and visually. For the visual comparison, Figure **2.15** shows the enlarged selected region of each image shown in Figure **2.14**.

We set up a second controlled simulated experiment by considering JPEG compression (blocking) artifacts which result from compression of the pepper image (256×256) shown in Figure **2.16**(a). The JPEG images was constructed by MATLAB's JPEG compression routine with quality factor 10. This compressed image is shown in Figure **2.16**(b), and the resulting RMSE value is 9.76. We again applied several denoising methods (similar to the ones used in the previous example). The results of applying classic KR (1.35) with $N = 2$ and $h = 1.0$, bilateral filter (2.3) with $h_s = 2.0$ and $h_p = 4.1$, the anisotropic diffusion [25], the wavelet method [46], K-SVD [44], BM3D [45], and the iterative SKR with $N = 2$, $h = 2.0$, and 3 iterations are shown in Figure **2.16**(c)-(h), respectively. The corresponding RMSE values are (b)9.76, (c)9.03, (d)8.52, (e)9.70 (f)8.80, (g)8.54, (h)8.60, and (i)8.48.

The third experiment is a denoising example with the color JFK image (367×343) carrying real film grain noise and scanning process noise, shown in Figure **2.17**(a). To produce better color estimates, following [47], first we convert this RGB image to the YCbCr representation. Then, we applied K-SVD [44], BM3D [45], and iterative SKR on each channel (the luminance component Y, and the chrominance components Cb

[10]Root Mean Square Error is defined as $\frac{1}{\Omega}\|\underline{\mathbf{u}} - \hat{\underline{\mathbf{z}}}\|_2$, where Ω is the total number of pixels, $\underline{\mathbf{u}}$ is the true signal (image), and $\hat{\underline{\mathbf{z}}}$ is the estimated signal (image).



(a) Original Lena image

(b) Noisy image
RMSE = 25.0

(c) Classic KR (1.35)
RMSE = 8.94

(d) Bilateral filter [7]
RMSE = 8.65

(e) Anisotropic diffusion [25]
RMSE = 8.64

(f) Wavelet [46]
RMSE = 6.66

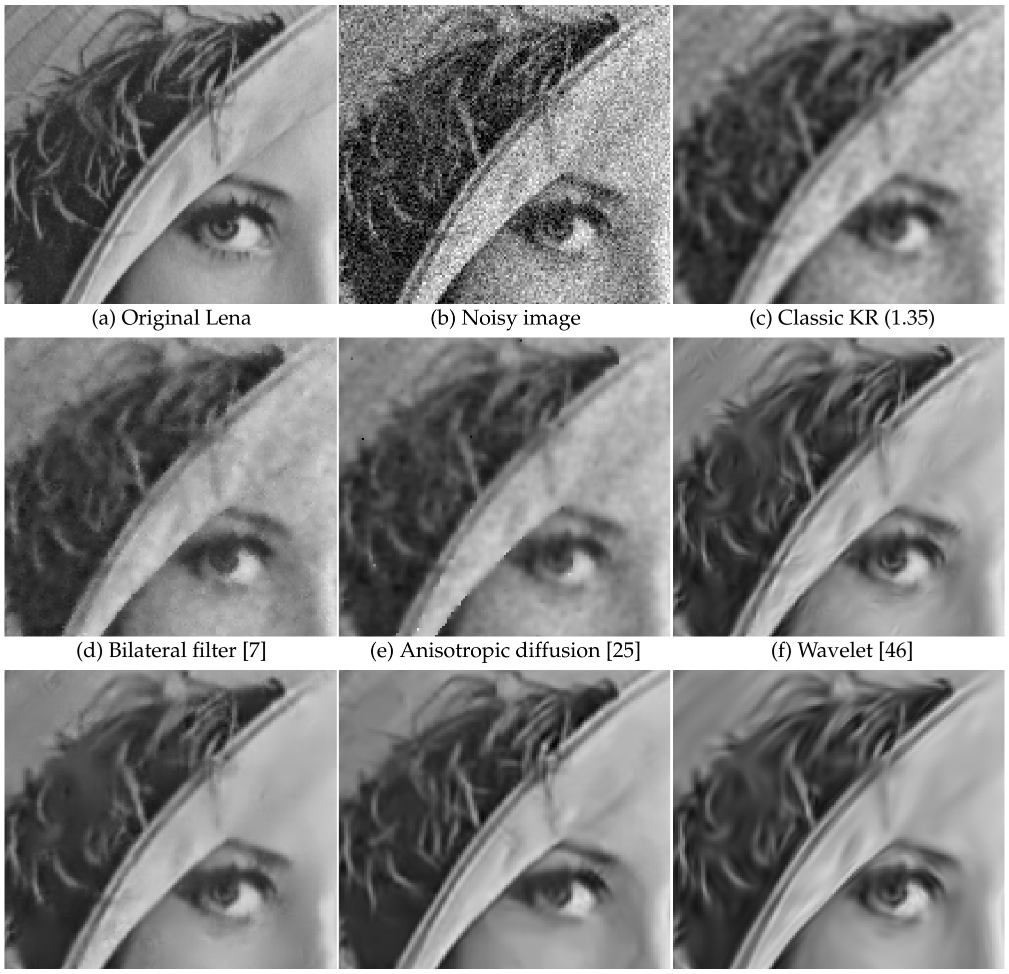
(g) K-SVD [44]
RMSE = 6.90

(h) BM3D [45]
RMSE = 6.35

(i) Iterative SKR
RMSE = 6.64

Figure 2.14: An example of Gaussian noise removal with the Lena image: (a) the original Lena image, (b) the noise-ridden image that we generated by adding white Gaussian noise with standard deviation 25 (SNR = 5.64[dB]), (c) the second order classic KR (1.35), (d) bilateral filter [7], (e) anisotropic diffusion [25], (f) a wavelet method [46], (g) K-SVD [44], and (i) Iterative SKR. The corresponding RMSE values are (b)25, (c)8.94, (d)8.65, (e)8.64, (f)6.66, (g)6.90, (h)6.35, and (i) 6.64.

and Cr), separately. The restored images by K-SVD, BM3D, and iterative SKR ($N = 2$, $h = 2.0$, and 3 iterations) are shown in Figures **2.17**(b)-(d), respectively. Figures **2.18**(b)-



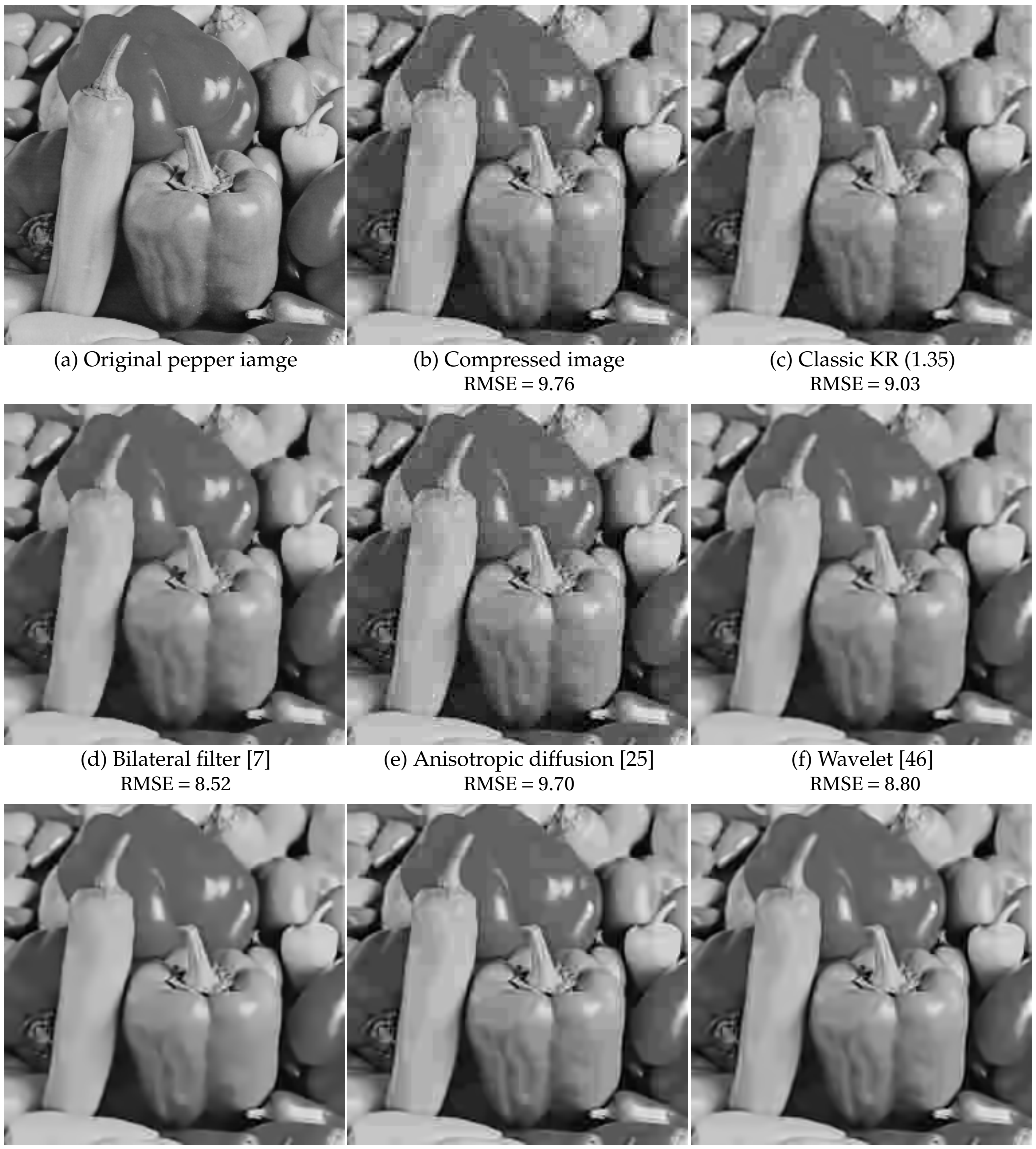
(a) Original Lena (b) Noisy image (c) Classic KR (1.35)

(d) Bilateral filter [7] (e) Anisotropic diffusion [25] (f) Wavelet [46]

(g) K-SVD [44] (h) BM3D [45] (i) Iterative SKR

Figure 2.15: An example of white Gaussian noise removal with the Lena image: the images (a)-(i) show the enlarged selected regions of the respective images shown in Figure **2.14**.

(d) show the absolute values of the residuals (the difference between the noise-ridden signal and the restored signal), on the Y component of K-SVD, BM3D, and iterative SKR, respectively.



(a) Original pepper iamge

(b) Compressed image
RMSE = 9.76

(c) Classic KR (1.35)
RMSE = 9.03

(d) Bilateral filter [7]
RMSE = 8.52

(e) Anisotropic diffusion [25]
RMSE = 9.70

(f) Wavelet [46]
RMSE = 8.80

(g) K-SVD [44]
RMSE = 8.54

(h) BM3D [45]
RMSE = 8.60

(i) Iterative SKR
RMSE = 8.48

Figure 2.16: An example of blocking artifact reduction with the pepper image: (a) the original pepper image, (b) the compressed image that we generated by compressing with the quality factor 10, (c) the second order classic KR (1.35), (d) bilateral filter [7], (e) Anisotropic diffusion [25], (f) a wavelet method [46], (g) K-SVD [44], (h) BM3D [45], and (i) Iterative steering KR. The corresponding RMSE values are (b)9.76, (c)9.03, (d)8.52, (e)9.70, (f)8.80, (g)8.54, (h)8.60, and (i)8.48.

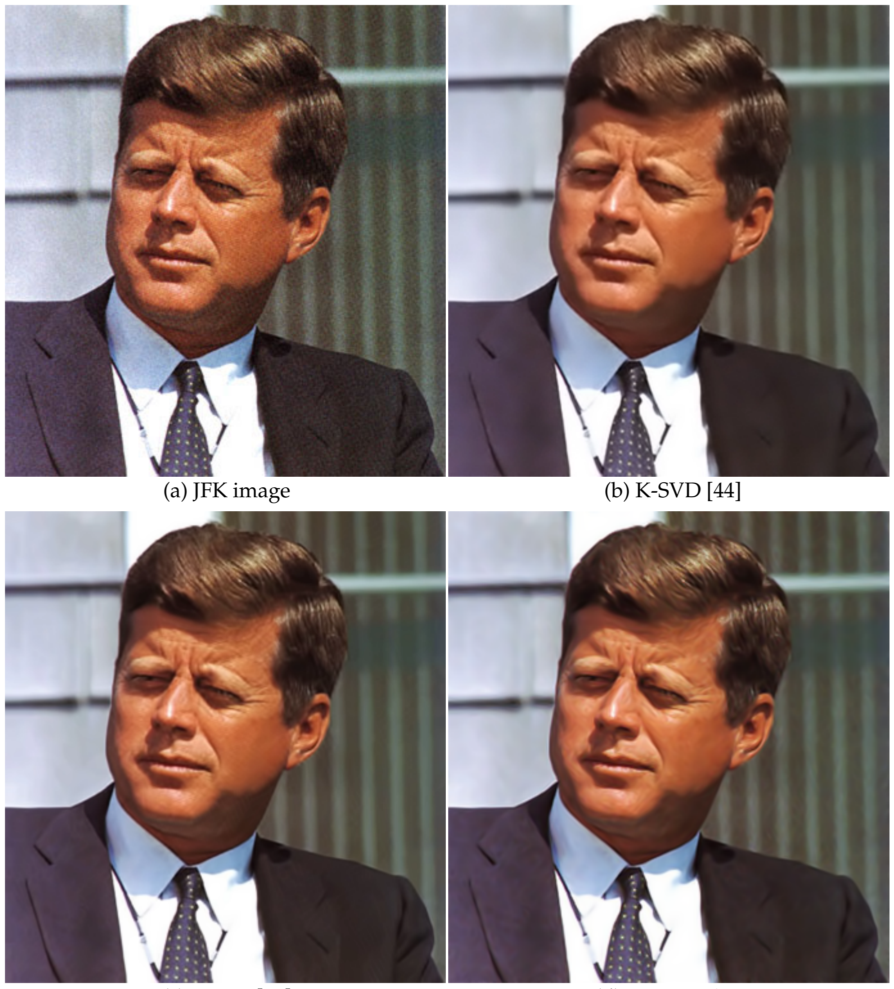

(a) JFK image (b) K-SVD [44] (c) BM3D [45] (d) Iterative SKR

Figure 2.17: An example of film grain removal: (a) the JFK image that carries real film grain noise, (b) K-SVD [44], (c) BM3D [45], and (d) iterative SKR.

2.2.2 Image Reconstruction from Irregularly Sampled Data

The fourth and fifth experiments are controlled simulated interpolations of irregularly sampled images: house (256×256) and Lena (512×512). We randomly deleted 85% of the pixels in the House image and the Lena image of Figure **2.19**(b), and Figure **2.20**(b), generating the irregularly sampled images shown in Figure **2.19**(a) and Figure **2.20**(a), in which the black regions represent missing pixels. To fill the miss-

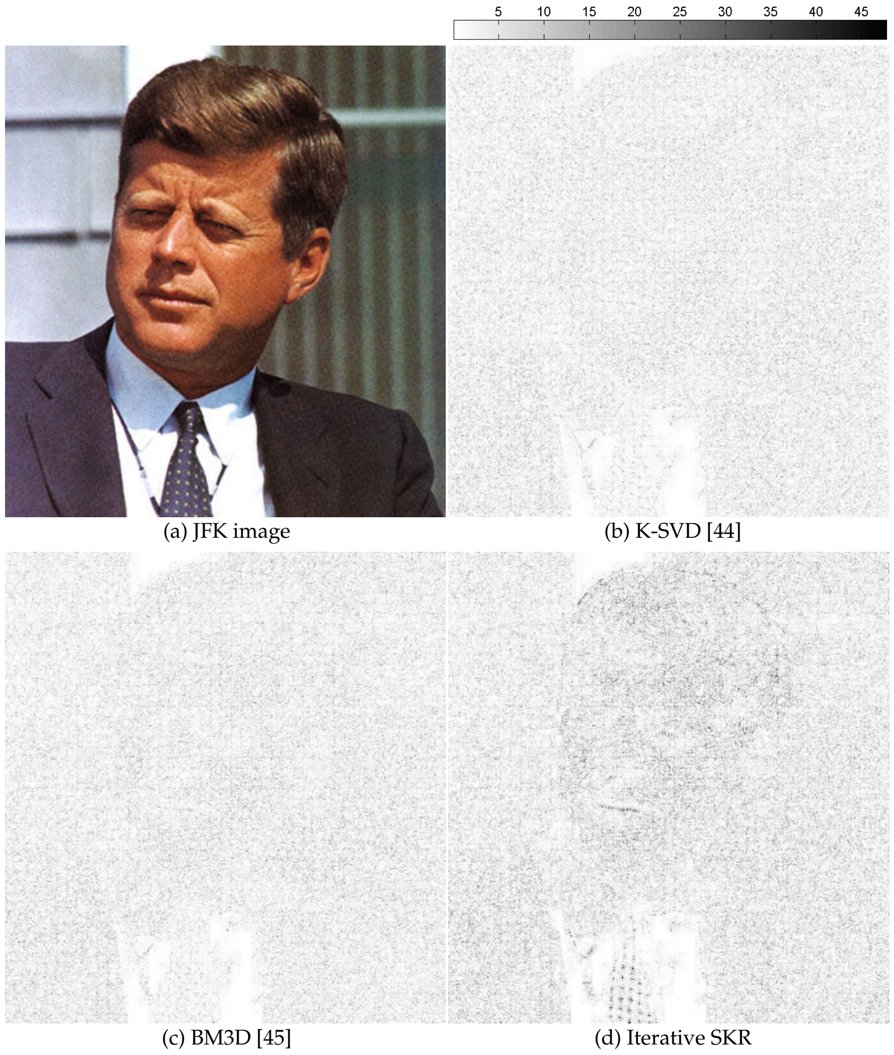
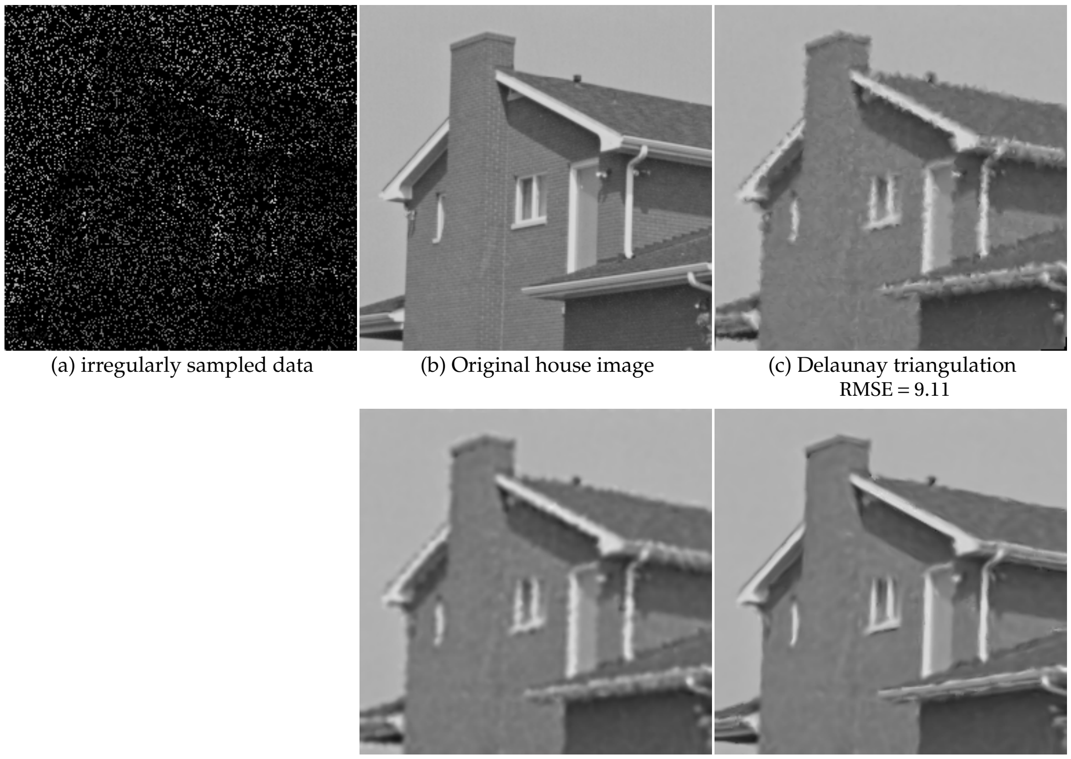


Figure 2.18: The comparison of the film grain removal on the JFK image by residual images (the absolute differences between the JFK image and the denoised images): (a) the JFK image that carries real film grain noise, (b) K-SVD [44], (c) BM3D [45], and (d) iterative SKR.

ing values, we first applied Delaunay triangulation[11], and the reconstructed image is shown in Figure **2.19**(c) and Figure **2.20**(c). Next, we applied classic KR (1.35) with $N = 2$ and $h = 2.25$ and iterative SKR with $N = 2$, $h = 1.6$, and 1 iteration), and the results are shown in Figures **2.19**(c)-(d) and Figures **2.20**(c)-(d), respectively. The corre-

[11]We used MATLAB's routine "griddata" function with "linear" method.



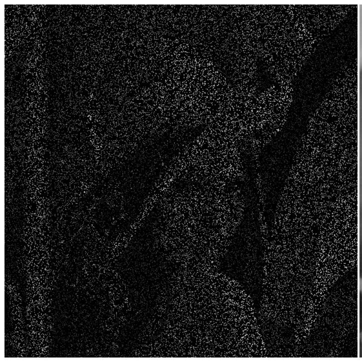
(a) irregularly sampled data (b) Original house image (c) Delaunay triangulation RMSE = 9.11

(d) Classic KR (1.35) RMSE = 10.36 (e) Iterative SKR RMSE = 8.72

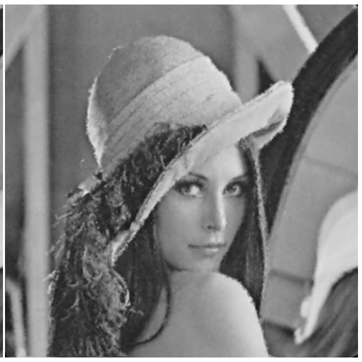
Figure 2.19: An example of image reconstruction from irregularly sampled data with the house image: (a) the irregularly sampled data that we generated by randomly eliminating 85% of pixels (no noise), (b) the original house image, (c) Delaunay triangulation, (d) classic KR (1.35), and (e) Iterative SKR. The corresponding RMSE values of the reconstructed images are (c)9.11, (d)10.36, and (e)8.72.

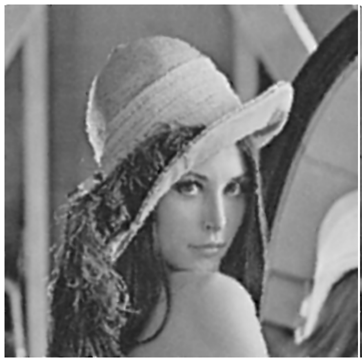
sponding RMSE values for the House experiment are (c)9.11, (d)10.36, and (e)8.72, and (c)9.29, (d)9.72, and (e)8.21 for the Lena experiment.

The final experiment is a multi-frame resolution enhancement (also known as *super-resolution* [2]) of a real compressed color image sequence captured with a commercial video surveillance camera, courtesy of Adyoron Intelligence System, Ltd., Tel Aviv, Israel. We cropped the original input Adyoron sequence into 100×100 region with 10 frames, where the first frame is shown in Figure **2.21**(a). Again, to produce bet-


(a) irregularly sampled data

(b) Original Lena image


(c) Delaunay triangulation
RMSE = 9.29


(d) Classic KR (1.35)
RMSE = 9.72

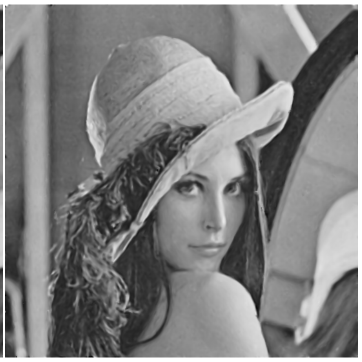
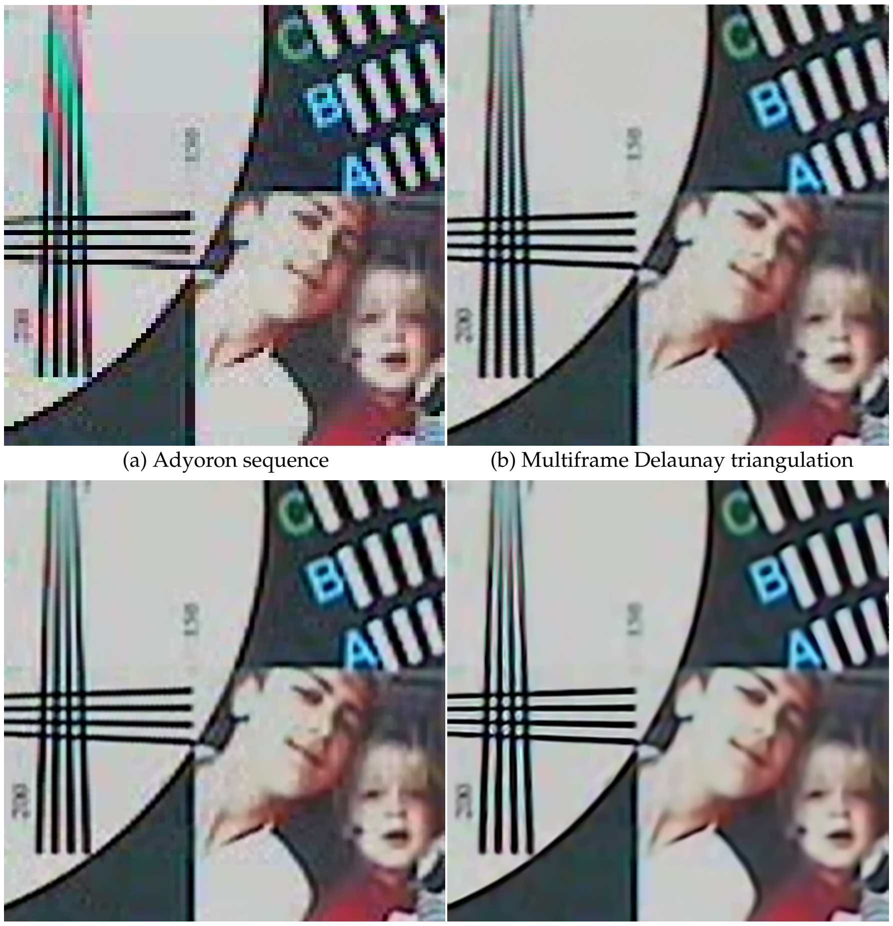

(e) Iterative SKR
RMSE = 8.21

Figure 2.20: An example of image reconstruction from irregularly sampled data with the Lena image: (a) the irregularly sampled data that we generated by randomly eliminating 85% of pixels (no noise), (b) the original house image, (c) Delaunay triangulation, (d) classic KR (1.35), and (e) Iterative SKR. The corresponding RMSE values of the reconstructed images are (c)9.29, (d)9.72, and (e)8.21.

ter color estimates, following [47], we convert the RGB video frames into the YCbCr representation, and processed each channel separately. Assuming that the underlying motion is translational (q.v. Appendix D), we estimate motions between frames of the Y channel, and obtain an irregularly sampled data set as described in Figure **1.3** for each channel. In this example, we set the resolution enhancement factor of 1:5 (i.e. 25 times as many overall pixels), reconstruct high-resolution images from the irregular data by Delaunay triangulation, classic KR (1.35) ($N = 2$ and $h = 2.0$ for the Y


(a) Adyoron sequence (b) Multiframe Delaunay triangulation

(c) Multiframe classic KR (1.35) (d) Multiframe iterative SKR

Figure 2.21: An example of resolution enhancement with the Adyoron video sequence: (a) the first video frame of the Adyoron sequence, (b) multiframe Delaunay triangulation, (c) multiframe classic KR (1.35), and (d) multiframe iterative SKR.

channel, $h = 3.5$ for the Cb and Cr channels), and iterative SKR ($N = 2$, $h = 4.0$ for the Y channel, $h = 8.0$ for the Cb and Cr channels, and 1 iteration), and then deblurred the reconstructed images using Bilateral total variation regularization[12] [2]. The final results using Delaunay triangulation, classic KR, and iterative SKR are shown in Figures **2.21**(b)-(d), respectively. The iterative SKR successfully enhances the resolution

[12]For this experiment, the camera point spread function (PSF) was assumed to be a 5×5 disk kernel (obtained by the MATLAB's routine "fspecial" with "disk" option and the radius parameter of 2.

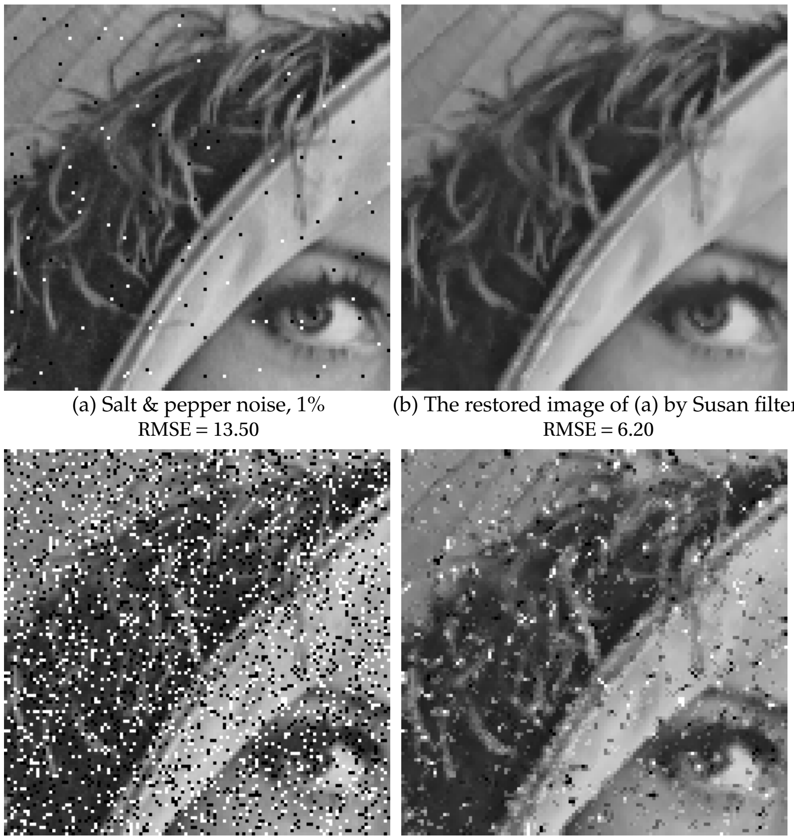
while suppressing compression artifacts effectively.

2.3 Robust Kernel Regression

As we briefly discussed in Section 2.1.1, when the given image is contaminated by a few outliers, such as salt & pepper noise, Susan filter (2.6) is a possible method to eliminate them. An example of salt & pepper noise reduction using Susan filter is shown in Figure **2.22**. As seen in the results, Susan filter works well only for a few outliers. When the number of outliers increases, some outliers are too close to each other, and leaving out the center pixel only is no longer effective. Hence we need an alternative approach. Specifically, in this section, we introduce the technique of *robust estimator* [48, 49] into the kernel regression framework.

So far, in the data models (1.1) and (1.17), we assume that ε_i is zero-mean *i.i.d.* noise, such as white Gaussian noise, and, therefore, the weighted least square (weighted L_2-norm) was a good choice to remove noise while preserving major image structures as shown in Section 2.2. Although such limiting assumptions facilitate the design of optimal methods for a certain type of data, in real situations when the data and noise models do not faithfully describe the measured signals, the performance of such non-robust methods significantly degrades [2].

The robust estimation technique explicitly reflects the statistical property of noise in the optimization. To be specific, when the noise statistics are more heavy-tailed, the squared error between the given data y_i and the estimate $\hat{z}(\cdot)$ (i.e. $(y_i - \beta_0 - \boldsymbol{\beta}_1^T(\mathbf{x}_i - \mathbf{x}) - \cdots)^2$) is no longer a suitable choice. Thus, we modify the optimization of



(a) Salt & pepper noise, 1%
RMSE = 13.50

(b) The restored image of (a) by Susan filter
RMSE = 6.20

(c) Salt & pepper noise, 20%
RMSE = 63.84

(d) The restored image of (c) by Susan filter
RMSE = 28.33

Figure 2.22: An example of salt & pepper noise reduction by Susan filter (2.6): (a),(c) noise-ridden images that we generated by adding 1% and 20% salt & pepper noise, respectively, and (b),(d) the restored images of (a) and (c) by Susan filter, respectively. The corresponding RMSE values are (a)13.50, (b)6.20, (c)63.84, and (d)28.33.

kernel regression (2.1) with the generic *error norm function* $\varphi(\cdot)$ as

$$\min_{\{\boldsymbol{\beta}_n\}} \sum_{i=1}^{P} \varphi\Big(y_i - \beta_0 - \boldsymbol{\beta}_1^T(\mathbf{x}_i - \mathbf{x}) - \boldsymbol{\beta}_2^T \text{vech}\big\{(\mathbf{x}_i - \mathbf{x})(\mathbf{x}_i - \mathbf{x})^T\big\} - \cdots\Big) K_{\text{adapt}}(\mathbf{x}_i - \mathbf{x}, y_i - y). \quad (2.19)$$

The estimator above not takes not only the statistics of the underlying signal (image) into account, but the noise statistics are also taken into account by the data-adapted kernel $K_{\text{adapt}}(\cdot)$ and the error norm function $\varphi(\cdot)$, respectively. Some possible choices for $\varphi(\cdot)$ can be found in [50]. The robustness with respect to outliers (such as salt & pepper noise) can be significantly improved by exploiting the absolute error (i.e. $\varphi(\cdot) = |\cdot|$), which in effect incorporates L_1-norm estimator [2] in the kernel regression framework. Using the matrix form, the optimization problem (2.19) can be posed as weighted L_1-norm:

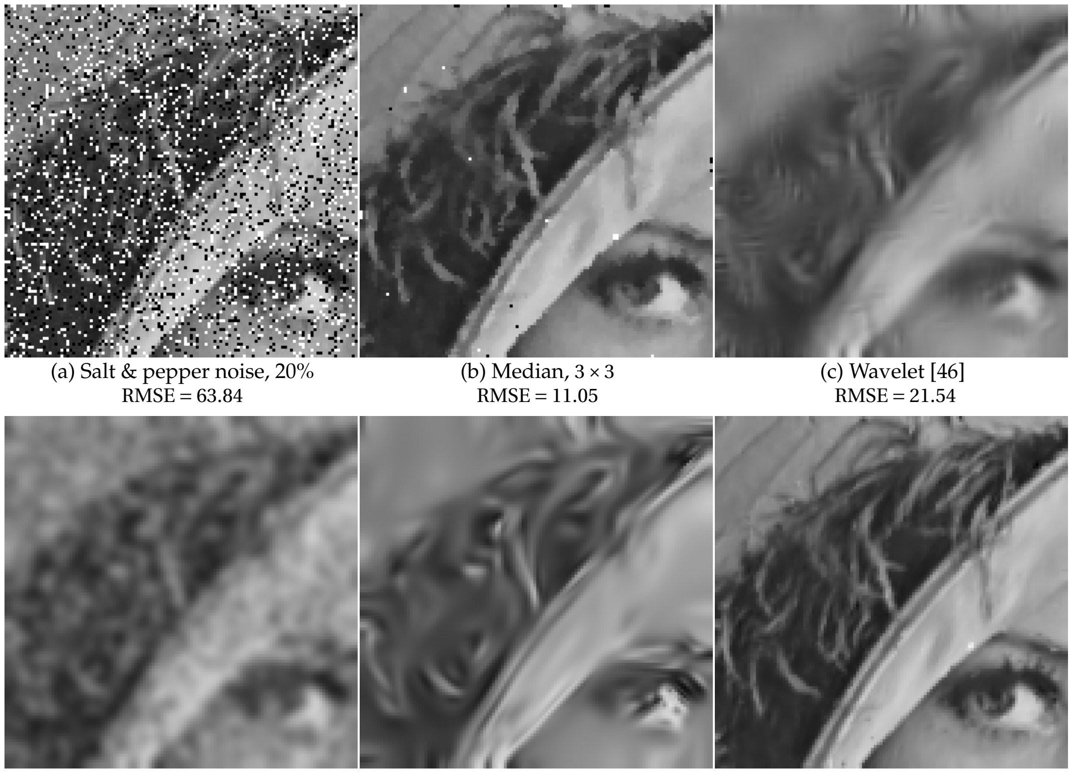
$$\widehat{\mathbf{b}} = \arg\min_{\mathbf{b}} \left\| \mathbf{K}\left(\mathbf{y} - \mathbf{X}\mathbf{b}\right) \right\|_1, \tag{2.20}$$

and we find the solution of the optimization by the steepest descent [28]:

$$\widehat{\mathbf{b}}^{(\ell+1)} = \widehat{\mathbf{b}}^{(\ell)} + \mu \mathbf{X}^T \mathbf{K} \, \mathrm{sign}\left(\mathbf{y} - \mathbf{X}\widehat{\mathbf{b}}^{(\ell)}\right), \tag{2.21}$$

where μ is a scalar defining the step size in the direction of the gradient.

Its implementation is similar to iterative steering kernel regression. In the diagram shown in Figure **2.12**, we replace the classic kernel gradient estimator (B.2) by (2.21) with classic kernel matrix (i.e. $\mathbf{K} = \mathrm{diag}\{K_{\mathbf{H}}(\mathbf{x}_1 - \mathbf{x}), \cdots, K_{\mathbf{H}}(\mathbf{x}_p - \mathbf{x})\}$), and then we compute steering matrices. Also replacing (2.12) by (2.21) with steering kernel matrix (i.e. $\mathbf{K} = \mathrm{diag}\{K_{\mathbf{H}_1^{\mathrm{steer}}}(\mathbf{x}_1 - \mathbf{x}), \cdots, K_{\mathbf{H}_P^{\mathrm{steer}}}(\mathbf{x}_p - \mathbf{x})\}$), we estimate the image $z(\cdot)$ again. Figure **2.23** shows an example of salt & pepper noise removal. In this experiment, we again added 20% salt & pepper noise to a part of Lena image, resulting in the noisy image of Figure **2.23**(a). Using (b) a 3×3 median filter, (c) wavelet method of [46], (d) L_2 classic KR (1.35) ($N = 2$ and $h = 2.46$), (e) L_2 iterative SKR (2.12) ($N = 2$, $h = 2.25$, and 20 iterations), and (f) L_1 iterative SKR ($N = 2$, $h = 2.25$, and 1 iteration), we denoised the noisy image. The corresponding RMSE values are (a)63.84, (b)11.05, (c)21.54, (d)21.81, (e)21.06, and (f)7.14, respectively. The example show that, while the wavelet method, L_2 classic KR, and L_2 steering KR are ineffective for salt & pepper removal, the L_1 steering KR removes the noise successfully and the result is much better than the median filter.


(a) Salt & pepper noise, 20%
RMSE = 63.84

(b) Median, 3×3
RMSE = 11.05

(c) Wavelet [46]
RMSE = 21.54

(d) L_2 Classic KR (1.35)
RMSE = 21.81

(e) L_2 iterative SKR (2.12)
RMSE = 21.06

(f) L_1 iterative SKR
RMSE = 7.14

Figure 2.23: An example of Salt & pepper noise reduction with a cropped Lena image: (a) a noise-ridden image that we generated by adding 20% salt & pepper noise, and (b) the restored image by 3×3 median filter, (c) a wavelet method [46], (d) L_2 classic KR (1.35), (e) L_2 iterative SKR (2.12), and L_1 iterative SKR. The corresponding RMSE values are (a)63.84, (b)11.05, (c)21.54, (d)21.81, (e)21.06, and (f)7.14, respectively.

Summary— In this chapter, in order to overcome the inherent limitations dictated by the locally linear filtering properties of the classic kernel regression framework, we developed the nonlinear data-adaptive class of kernel regressors. We showed that the popular bilateral filtering technique is a special case of adaptive regression. We introduced and justified a novel adaptive kernel regression method, called steering kernel with comparable or better performance with respect to state of the art methods. Moreover, we presented an iterative scheme to further improve the performance of the steering kernel regression method. An automatic method for picking the optimal smoothing parameter and the number of iterations as well as the optimal regression order is one of our important future works, which we include in Chapter 6.

Chapter 3

Kernel-Based Image Deblurring

Abstract— *Kernel regression* is an effective tool for a variety of image processing tasks, such as denoising and interpolation, as we described in Chapter 2. In this chapter, we extend the use of kernel regression for deblurring applications. In some earlier examples in the literature, such non-parametric deblurring was sub-optimally performed in two sequential steps, namely, denoising followed by deblurring. In contrast, our optimal solution jointly denoises and deblurs images. The kernel-based approach takes advantage of an effective and novel image prior that generalizes some of the most popular regularization techniques in the literature. Experimental results demonstrate the effectiveness of the method.

3.1 Introduction

The *kernel regression* framework [3] has been widely used in different guises for solving a variety of pattern detection and discrimination problems [4]. In Chapter 2, we described kernel regression as an effective tool for image restoration and re-

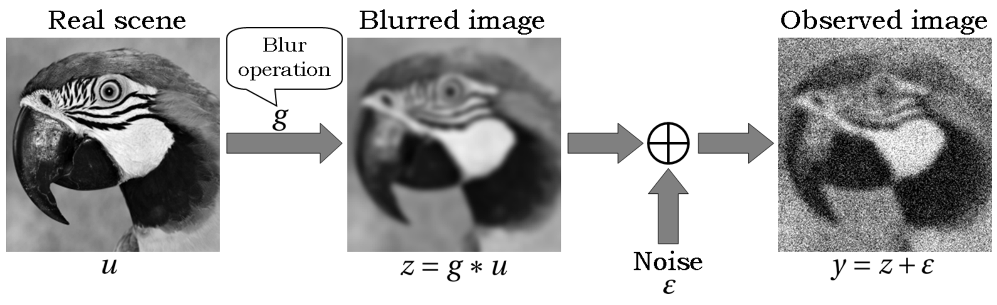


Figure 3.1: The data model for the deblurring problem.

construction, and established its relation with some popular existing techniques such as *normalized convolution* [51, 6], *bilateral filter* [7, 8], *edge-directed interpolation* [9], and *moving least-squares* [10]. Moreover, we proposed a novel adaptive generalization of kernel regression with excellent results in both denoising and interpolating (for single and multi-frame) applications. The image degradation model for all the above techniques mainly considered regularly or irregularly sampled data, contaminated with independent and identically distributed (*i.i.d*) additive zero mean noise (with otherwise no particular statistical distribution assumed).

In the previous chapter, an important image degradation source, namely the (out of focus or atmospheric) blur [2], was ignored. In other works, this problem was treated in a two-step process, for instance, denoising or interpolation followed by deblurring [52, 53]. Such two-step solutions in general are suboptimal, and improvements upon them are the subject of this chapter.

Generally, we have a model for the deblurring problem in Figure **3.1**. where u is the real scene which is the unknown function of interest, g is the point spread function (PSF) of the blur, ε is an additive white noise, and y is an observation that is the distorted function of u. Following this model, we express the pixel value y_i

measured at $\mathbf{x}_i$ by extending the data model (1.17) as

$$y_i = z(\mathbf{x}_i) + \varepsilon_i = (g * u)(\mathbf{x}_i) + \varepsilon_i, \tag{3.1}$$

where $*$ is the convolution operator, $z(\mathbf{x}_i) = (g * u)(\mathbf{x}_i)$ is the unknown blurred pixel value at a sampling position $\mathbf{x}_i$. In matrix notation, we write the blur-free pixels of interest as $\underline{\mathbf{u}} = [\cdots, u(\mathbf{x}_i), \cdots]^T$. Next, we rewrite the model (3.1) in matrix form as:

$$\underline{\mathbf{y}} = \mathbf{G}\underline{\mathbf{u}} + \underline{\boldsymbol{\varepsilon}}, \tag{3.2}$$

where $\mathbf{y} \in \mathscr{R}^{L \times M}$ is an observed (blurry and noisy) image, $\mathbf{u} \in \mathscr{R}^{L \times M}$ is the unknown image of interest, $\mathbf{G} \in \mathscr{R}^{LM \times LM}$ is the blur operation, and $\boldsymbol{\varepsilon} \in \mathscr{R}^{L \times M}$ is a noise image. The underline indicates that the matrices are lexicographically ordered into a column-stack vector, e.g., $\underline{\mathbf{u}} \in \mathscr{R}^{LM \times 1}$. Since the deblurring problem is typically ill-posed, a popular way to estimate the unknown image $\mathbf{u}$ is to use the regularized least square technique [29]:

$$\underline{\hat{\mathbf{u}}}_{\text{RLS}} = \arg\min_{\underline{\mathbf{u}}} \left\| \underline{\mathbf{y}} - \mathbf{G}\underline{\mathbf{u}} \right\|_2^2 + \lambda C_{\text{R}}(\underline{\mathbf{u}}), \tag{3.3}$$

where λ is the regularization parameter, and $C_{\text{R}}(\underline{\mathbf{u}})$ is the regularization term. Some representative examples of $C_{\text{R}}(\underline{\mathbf{u}})$ are

i. Tikhonov [54]

$$C_{\text{R}}(\underline{\mathbf{u}}) = \left\| \boldsymbol{\Gamma}\underline{\mathbf{u}} \right\|_2^2 \tag{3.4}$$

ii. Total variation (TV) [55]

$$C_{\text{R}}(\underline{\mathbf{u}}) = \left\| \boldsymbol{\Gamma}\underline{\mathbf{u}} \right\|_1 \tag{3.5}$$

iii. Bilateral total variation (BTV) [2]

$$C_{\text{R}}(\underline{\mathbf{u}}) = \sum_{l=-\zeta}^{\zeta} \sum_{m=-\zeta}^{\zeta} \eta^{|l|+|m|} \left\| \underline{\mathbf{u}} - \mathbf{S}_{x_1}^{l} \mathbf{S}_{x_2}^{m} \underline{\mathbf{u}} \right\|_1 \tag{3.6}$$

where $\mathbf{\Gamma} \in \mathscr{R}^{LM \times LM}$ is a high-pass filter, such as Laplacian [29], η is a smoothing parameter, ζ is the local analysis window size, and $\mathbf{S}_{x_1}^{l}, \mathbf{S}_{x_2}^{m} \in \mathscr{R}^{LM \times LM}$ are the shift operators that shift the image ($\underline{\mathbf{u}}$) l and m pixels in the x_1- and x_2-directions, respectively. In this chapter, we replace the likelihood term ($\|\underline{\mathbf{y}} - \mathbf{G}\underline{\mathbf{u}}\|_2^2$) in (3.3) with one motivated by the kernel regression technique, which results in a spatially adaptive weighted least square problem. Additionally, we introduce novel regularization terms which provide spatially adaptive prior information, resulting in a powerful overall setting for deconvolution.

Contributions of this chapter are the following: i) We describe and propose kernel regression as an effective tool for deblurring, ii) We also propose a novel image prior which is effective in suppressing both noise and ringing effects. These effects are typical in deblurring applications. We further show that many popular regularization techniques such as digital total variation (TV) [55], bilateral TV [2], and Tikhonov regularization [54] are special cases of this adaptive prior.

This chapter is organized as follows: In Section 3.2, we bring the blurring effect into the kernel regression (KR) framework, and introduce a regularized estimator based on the KR framework for the deblurring application. Also, we discuss the relationships with the existing approaches (i.e. Tikhonov, TV, and BTV). Next, we describe how to apply the data-adaptive kernels presented in Chapter 2 for the kernel-based deblurring estimator. Experiments are presented in Section 3.2.3. Furthermore, the kernel-based approach is applicable to other image restoration problems besides deblurring. We discuss the generalization of the kernel-based approach in Section 3.3.

3.2 Kernel-Based Deblurring

Deblurring is an ill-posed problem when the PSF completely kills high frequency components of the underlying image, and the ill-posedness of the deblurring problem causes the ringing and noise artifacts. An effective approach to reduce the undesirable effects is the regularization approach (3.3). In this section, first we will derive a likelihood term and a regularization term based on the kernel regression framework, and formulate a kernel-based deblurring estimator. As a result, the deblurring estimator becomes a regularized method with a choice of the kernel function. Using the data-adaptive kernel functions described in Chapter 2, we control the smoothness of the output image locally. Briefly speaking, the proposed deblurring estimator with the data-adaptive kernels recovers the high frequency components at the regions where edges and textures are present and smooths the regions where no image structures are present. We describe the details of the implementation of the kernel-based deblurring with the steering kernel function (2.9) in Section 3.2.2 and show some deblurring examples with comparisons to the existing deblurring methods in Section 3.2.3.

3.2.1 Kernel-Based Deblurring Estimator

Considering the blurring effect, instead of estimating the unknown function z as we have explained in Chapters 1 and 2, the function u is the one we wish to estimate. Therefore, in place of representing the blurred function z by a local approximation (Tayler series), we apply the kernel framework to the function u of interest and use the local signal representation between two unknown values, $u(\mathbf{x}_i)$ at a sampling position

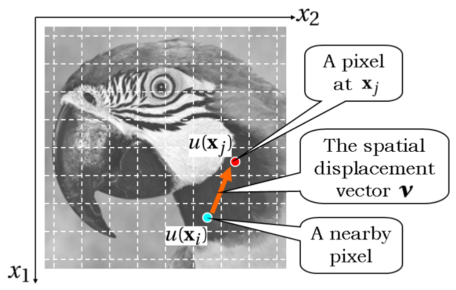

x_2
x_1
A pixel at $\mathbf{x}_j$
$u(\mathbf{x}_j)$
The spatial displacement vector $\boldsymbol{v}$
$u(\mathbf{x}_i)$
A nearby pixel

Figure 3.2: The i-th pixel at $\mathbf{x}_i$ and its neighboring pixel $u(\mathbf{x}_j)$ at $\mathbf{x}_i$ located v_1- and v_2-pixels away in x_1- and x_2-directions ($\boldsymbol{v} = [v_1, v_2]^T$), respectively.

$\mathbf{x}_i$ and $u(\mathbf{x})$ at a neighboring arbitrary position $\mathbf{x}$, with a Taylor series:

$$u(\mathbf{x}_i) = u(\mathbf{x}) + \{\nabla u(\mathbf{x})\}^T (\mathbf{x}_i - \mathbf{x}) + \frac{1}{2}(\mathbf{x}_i - \mathbf{x})^T \{\mathscr{H} u(\mathbf{x})\} (\mathbf{x}_i - \mathbf{x}) + \cdots . \tag{3.7}$$

Furthermore, unlike the denoising/interpolation problem where we use the noise only model (1.17) and derive a pointwise estimator (such as (2.12)), in the presence of blur, the blurring effect associates (couples) all the sampling positions $\mathbf{x}_i$'s with their neighbors, and this linkage precludes individual estimations of deblurred pixels, and necessitates a simultaneous estimation of all the pixels of interest (e.g. the regularization approach (3.3)). Hence we tightly constrain the reciprocal relationship between neighboring pixel values by a local representation model, i.e., the Taylor series, in order to derive a simultaneous pixel estimator based on the kernel regression framework. Suppose we have a pixel value $u(\mathbf{x}_j)$ and its neighbor $u(\mathbf{x}_i)$ which is located v_1-pixels in the x_1-direction and v_2-pixels in the x_2-direction away from $u(\mathbf{x}_i)$ as illustrated in Figure **3.2**. That is to say, we have

$$u(\mathbf{x}_i) = u(\mathbf{x}_j - \boldsymbol{v}), \tag{3.8}$$

where $\boldsymbol{v} = [v_1, v_2]^T$ is the spatial displacement vector. The Taylor series (3.7) indicates the following relationship between $u(\mathbf{x}_i)$ and $u(\mathbf{x}_j)$:

$$\begin{aligned} u(\mathbf{x}_i) &= u(\mathbf{x}_j - \boldsymbol{v}) \\ &= u(\mathbf{x}_j) + \left\{\nabla u(\mathbf{x}_j)\right\}^T \boldsymbol{v} + \frac{1}{2}\boldsymbol{v}^T\{\mathcal{H}u(\mathbf{x})\}\,\boldsymbol{v} + \cdots \\ &= u(\mathbf{x}_j) + \begin{bmatrix} u_{x_1}(\mathbf{x}_j) \\ u_{x_2}(\mathbf{x}_j) \end{bmatrix}^T \boldsymbol{v} + \frac{1}{2}\begin{bmatrix} u_{x_1^2}(\mathbf{x}_j) \\ 2u_{x_1x_2}(\mathbf{x}_j) \\ u_{x_2^2}(\mathbf{x}_j) \end{bmatrix}^T \operatorname{vech}\left\{\boldsymbol{v}\boldsymbol{v}^T\right\} + \cdots . \end{aligned} \tag{3.9}$$

In order to estimate all the pixels simultaneously as an image, we write the local representation (3.9) at every sampling point ($\mathbf{x}_i$'s) together, and gather the result into lexicographically stacked vector form as

$$\begin{aligned} \begin{bmatrix} \vdots \\ u(\mathbf{x}_j - \boldsymbol{v}) \\ \vdots \end{bmatrix} &= \begin{bmatrix} \vdots \\ u(\mathbf{x}_j) \\ \vdots \end{bmatrix} + \begin{bmatrix} \vdots \\ u_{x_1}(\mathbf{x}_j) \\ \vdots \end{bmatrix} v_1 + \begin{bmatrix} \vdots \\ u_{x_2}(\mathbf{x}_j) \\ \vdots \end{bmatrix} v_2 \\ &\quad + \frac{1}{2}\begin{bmatrix} \vdots \\ u_{x_1^2}(\mathbf{x}_j) \\ \vdots \end{bmatrix} v_1^2 + \begin{bmatrix} \vdots \\ u_{x_1x_2}(\mathbf{x}_j) \\ \vdots \end{bmatrix} v_1v_2 + \frac{1}{2}\begin{bmatrix} \vdots \\ u_{x_2^2}(\mathbf{x}_j) \\ \vdots \end{bmatrix} v_2^2 + \cdots \end{aligned} \tag{3.10}$$

For convenience, we denote, for example, the first and the second right hand vectors in (3.10) as the lexicographically ordered desired image ($\underline{\mathbf{u}}$) and its first derivative along the x_1-direction ($\underline{\mathbf{u}}_{x_1}$), respectively. Additionally, the left-hand vector in (3.10) can be regarded as the shifted version of $\underline{\mathbf{u}}$, and thus, we simplify the notation in (3.10) as

$$\mathbf{S}_{x_1}^{-v_1}\mathbf{S}_{x_2}^{-v_2}\underline{\mathbf{u}} = \underline{\mathbf{u}} + \underline{\mathbf{u}}_{x_1}v_1 + \underline{\mathbf{u}}_{x_2}v_2 + \underline{\mathbf{u}}_{x_1^2}v_1^2 + \underline{\mathbf{u}}_{x_1x_2}v_1v_2 + \underline{\mathbf{u}}_{x_2^2}v_2^2 + \cdots , \tag{3.11}$$

where $\mathbf{S}_{x_1}^{-v_1}$ and $\mathbf{S}_{x_2}^{-v_2}$ are the shift operators that shift an image by $(-v_1)$ and $(-v_2)$ pixels along x_1- and x_2-directions, respectively, and we absorb the coefficient $1/2$ in $\underline{\mathbf{u}}_{x_1^2}$ and

$\underline{\mathbf{u}}_{x_2^2}$. Finally, considering an N-th order approximation, we have

$$\begin{aligned}
\underline{\mathbf{u}} &= \mathbf{S}_{x_1}^{v_1}\mathbf{S}_{x_2}^{v_2}\left(\underline{\mathbf{u}}+\underline{\mathbf{u}}_{x_1}v_1+\underline{\mathbf{u}}_{x_2}v_2+\underline{\mathbf{u}}_{x_1^2}v_1^2+\underline{\mathbf{u}}_{x_1x_2}v_1v_2+\underline{\mathbf{u}}_{x_2^2}v_2^2+\cdots\right)\\
&\approx \mathbf{S}_{x_1}^{v_1}\mathbf{S}_{x_2}^{v_2}\mathbb{I}_N\mathbb{U}_N, \qquad (3.12)
\end{aligned}$$

where, for example, when $N=2$, we have

$$\begin{aligned}
\mathbb{I}_2 &= \begin{bmatrix}\mathbf{I} & \mathbf{I}_{v_1} & \mathbf{I}_{v_2} & \mathbf{I}_{v_1^2} & \mathbf{I}_{v_1v_2} & \mathbf{I}_{v_2^2}\end{bmatrix}\\
\mathbb{U}_2 &= \begin{bmatrix}\underline{\mathbf{u}}^T & \underline{\mathbf{u}}_{x_1}^T & \underline{\mathbf{u}}_{x_2}^T & \underline{\mathbf{u}}_{x_1^2}^T & \underline{\mathbf{u}}_{x_1x_2}^T & \underline{\mathbf{u}}_{x_2^2}^T\end{bmatrix}^T \qquad (3.13)
\end{aligned}$$

where we define $\mathbf{I}_{v_1}=\operatorname{diag}\{v_1,\cdots,v_1\}$. Naturally, with fewer higher-order terms in this expansion, the smaller shift distances result in a more faithful approximation in (3.12). This suggests a general (smoothness) prior (regularization term) with weights given by the shift (spatial) vector $\boldsymbol{v}$ for images smooth to order N as

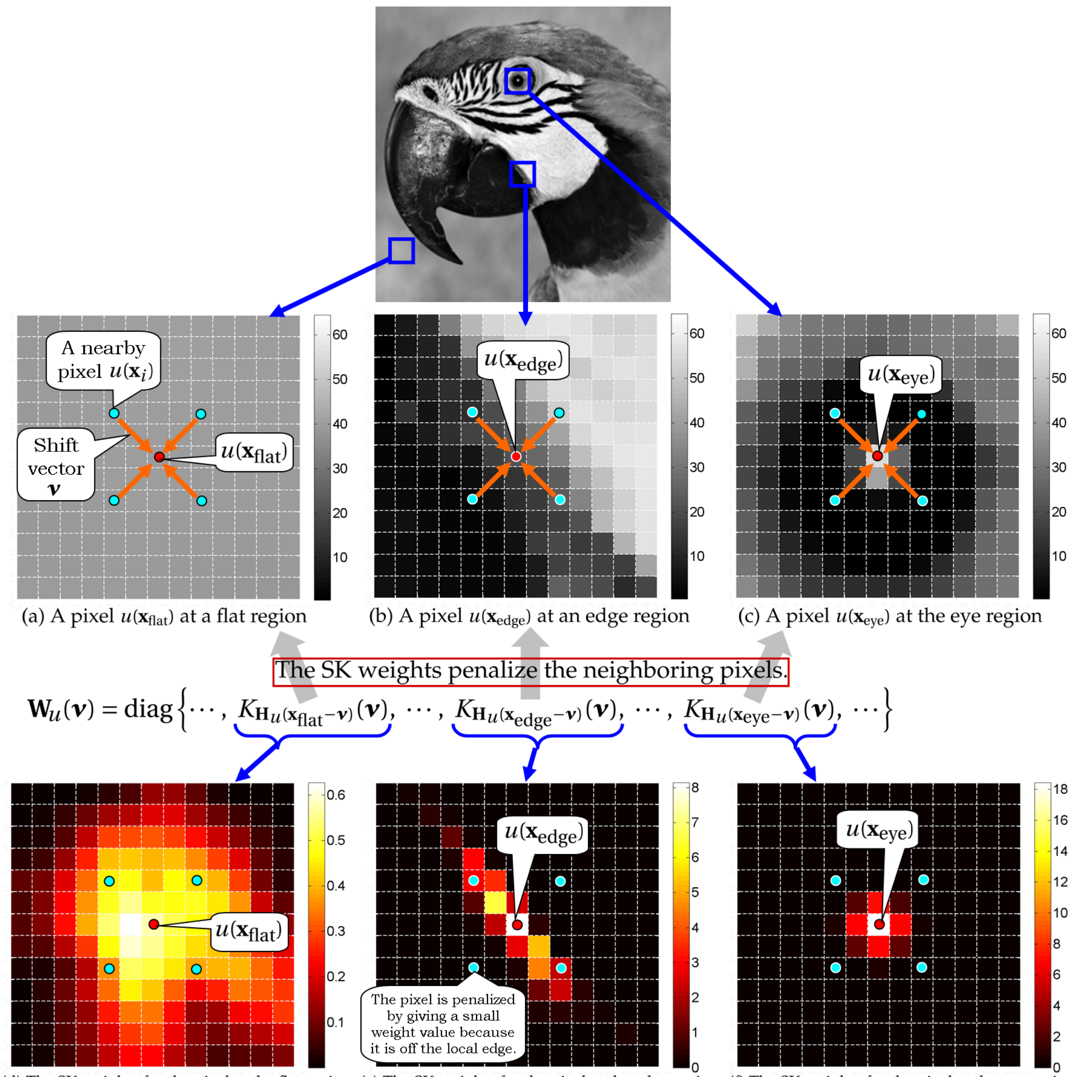
$$C_{\mathrm{R}}(\mathbb{U}_N)=\sum_{v_1=-\zeta}^{\zeta}\sum_{v_2=-\zeta}^{\zeta}\left\|\{\mathbf{W}_u(\boldsymbol{v})\}^{\frac{1}{q}}\left(\underline{\mathbf{u}}-\mathbf{S}_{x_1}^{v_1}\mathbf{S}_{x_2}^{v_2}\mathbb{I}_N\mathbb{U}_N\right)\right\|_q^q \qquad (3.14)$$

which we call the *adaptive kernel total variation* (AKTV) when $q=1$, and where $\mathbf{W}_u(\boldsymbol{v})$ is the weight matrix and a function of the shift vector $\boldsymbol{v}$ for this term defined as

$$\mathbf{W}_u(\boldsymbol{v})=\operatorname{diag}\left\{\cdots,\ K_{\mathbf{H}_u(\mathbf{x}_j-\boldsymbol{v})}(\boldsymbol{v}),\ \cdots\right\}. \qquad (3.15)$$

Here again, $K(\cdot)$ is the kernel function and $\mathbf{H}_u(\mathbf{x}_j-\boldsymbol{v})$ is the smoothing matrix at $(\mathbf{x}_j-\boldsymbol{v})$ computed based on the unknown (estimated) function $\hat{u}(\cdot)$ as described in detail in Section 3.2.2. The deblurring problem is often ill-posed, in particular, when the width of the PSF is large. Therefore, the regularization term (3.14) is useful to further restrict the solution space for the signal of interest. Figure **3.3** illustrates a graphical representation of the proposed regularization term. Figure **3.3**(a)-(c) show three pixels ($u(\mathbf{x}_{\text{flat}})$, $u(\mathbf{x}_{\text{edge}})$, and $u(\mathbf{x}_{\text{eye}})$) at a flat region, an edge region, and the eye with some of their neighboring pixels. Those neighboring pixels are shifted with the vector $\boldsymbol{v}$ in order to smooth the pixels ($u(\mathbf{x}_{\text{flat}})$, $u(\mathbf{x}_{\text{edge}})$, and $u(\mathbf{x}_{\text{eye}})$). When using the higher regression order (e.g. $N = 1$ or 2), the shift operation involves the derivatives, as described in (3.12). The derivatives compensate the pixel values based on the approximation by the Taylor expansion of the signal of interest, and, consequently, we estimate the underlying signal with the locally linear model ($N = 1$) and the locally quadratic model ($N = 2$). Furthermore, we penalize the neighboring pixels with the weights given by the data-adaptive (steering) kernel function, which we will describe in Section 3.2.2. The steering weight values for the neighboring pixels of $u(\mathbf{x}_{\text{flat}})$, $u(\mathbf{x}_{\text{edge}})$, and $u(\mathbf{x}_{\text{eye}})$ are shown in Figure **3.3**(d)-(f), respectively. The weights control how much the neighboring pixels should contribute to the smoothing operations locally. For instance, the steering kernel for the nearby pixels around the pixel at the flat region ($u(\mathbf{x}_{\text{flat}})$) spreads wider as shown in Figure **3.3**(d), and it indicates the pixel $u(\mathbf{x}_{\text{flat}})$ will be smoothed with all the neighbors. That is preferable for the flat region to remove noise. On the other hand, for the pixel at the edge region ($u(\mathbf{x}_{\text{edge}})$), we have the steering weights spreading along the local edge as shown in Figure **3.3**(e). Hence, our regularization term smooths the pixel $u(\mathbf{x}_{\text{edge}})$ with the neighboring pixels which belong to the same edge (i.e. the neighboring pixels at upper-left and lower-right), and it largely excludes the other pixels. In addition, the steering kernel stays small at the eye region and excludes all the neighboring pixel from the smoothing operation by giving low weights. Hence, the data-adaptive kernel approach is effective to suppress noise while preserving the image structures (edges and textures).

Having introduced the regularization term, we now turn our attention to describing the kernel-based data fidelity term. Using the local representation (3.12), the


A nearby pixel $u(\mathbf{x}_i)$
Shift vector $\boldsymbol{\nu}$
$u(\mathbf{x}_{\text{flat}})$
$u(\mathbf{x}_{\text{edge}})$
$u(\mathbf{x}_{\text{eye}})$
The SK weights penalize the neighboring pixels.
$\mathbf{W}_u(\boldsymbol{\nu}) = \text{diag}\left\{\cdots,\ \underbrace{K_{\mathbf{H}_u(\mathbf{x}_{\text{flat}}-\boldsymbol{\nu})}(\boldsymbol{\nu})},\ \cdots,\ \underbrace{K_{\mathbf{H}_u(\mathbf{x}_{\text{edge}}-\boldsymbol{\nu})}(\boldsymbol{\nu})},\ \cdots,\ \underbrace{K_{\mathbf{H}_u(\mathbf{x}_{\text{eye}}-\boldsymbol{\nu})}(\boldsymbol{\nu})},\ \cdots\right\}$
The pixel is penalized by giving a small weight value because it is off the local edge.

(a) A pixel $u(\mathbf{x}_{\text{flat}})$ at a flat region (b) A pixel $u(\mathbf{x}_{\text{edge}})$ at an edge region (c) A pixel $u(\mathbf{x}_{\text{eye}})$ at the eye region

(d) The SK weights for the pixel at the flat region (e) The SK weights for the pixel at the edge region (f) The SK weights for the pixel at the eye region

Figure 3.3: A graphical representation of the proposed regularization term (3.14): The figures (a)-(c) shows the pixels ($u(\mathbf{x}_{\text{flat}})$, $u(\mathbf{x}_{\text{edge}})$, and $u(\mathbf{x}_{\text{eye}})$) at a flat region, an edge region and the eye, respectively, with their neighboring pixels, and the figures (d)-(f) shows the steering kernel weight values given by $K_{\mathbf{H}_u(\mathbf{x}_j-\boldsymbol{\nu})}(\boldsymbol{\nu})$ for the pixels ($u(\mathbf{x}_{\text{flat}})$, $u(\mathbf{x}_{\text{edge}})$), and $u(\mathbf{x}_{\text{eye}})$).

blurred noisy image $\mathbf{y}$ is expressed as

$$\begin{aligned}\underline{\mathbf{y}} &= \mathbf{G}\underline{\mathbf{u}} + \underline{\boldsymbol{\varepsilon}} \\ &= \mathbf{G}\mathbf{S}_{x_1}^{\nu_1}\mathbf{S}_{x_2}^{\nu_2}\left(\underline{\mathbf{u}} + \underline{\mathbf{u}}_{x_1}\nu_1 + \underline{\mathbf{u}}_{x_2}\nu_2 + \underline{\mathbf{u}}_{x_1^2}\nu_1^2 + \underline{\mathbf{u}}_{x_1x_2}\nu_1\nu_2 + \underline{\mathbf{u}}_{x_2^2}\nu_2^2 + \cdots\right) + \underline{\boldsymbol{\varepsilon}} \\ &\approx \mathbf{S}_{x_1}^{\nu_1}\mathbf{S}_{x_2}^{\nu_2}\mathbb{G}_N\mathbb{U}_N + \underline{\boldsymbol{\varepsilon}} \qquad (3.16)\end{aligned}$$

where

$$\mathbb{G}_N = \begin{bmatrix} \mathbf{GI} & \mathbf{GI}_{v_1} & \mathbf{GI}_{v_2} & \mathbf{GI}_{v_1^2} & \mathbf{GI}_{v_1 v_2} & \mathbf{GI}_{v_2^2} \end{bmatrix}. \tag{3.17}$$

Similar to the regularization term (3.14), the local representation for the blurred noisy image (3.16) suggests a data fidelity or likelihood term:

$$C_{\mathrm{L}}(\mathbb{U}_N) = \sum_{v_1=-\zeta}^{\zeta} \sum_{v_2=-\zeta}^{\zeta} \left\| \{\mathbf{W}_z(\boldsymbol{v})\}^{\frac{1}{2}} \left(\underline{\mathbf{y}} - \mathbf{S}_{x_1}^{v_1} \mathbf{S}_{x_2}^{v_2} \mathbb{G}_N \mathbb{U}_N \right) \right\|_2^2 \tag{3.18}$$

where $\mathbf{W}_z(\boldsymbol{v})$ is the weight matrix for this likelihood term computed based on the estimated blurred signal $\hat{z} = g * \hat{u}$ as we will describe in Section 3.2.2.

In summary, the overall cost function to be minimized is formulated as

$$\begin{aligned} C(\mathbb{U}_N) &= C_{\mathrm{L}}(\mathbb{U}_N) + \lambda C_{\mathrm{R}}(\mathbb{U}_N) \\ &= \sum_{v_1=-\zeta}^{\zeta} \sum_{v_2=-\zeta}^{\zeta} \left[\left\| \{\mathbf{W}_z(\boldsymbol{v})\}^{\frac{1}{2}} \left(\underline{\mathbf{y}} - \mathbf{S}_{x_1}^{v_1} \mathbf{S}_{x_2}^{v_2} \mathbb{G}_N \mathbb{U}_N \right) \right\|_2^2 + \lambda \left\| \{\mathbf{W}_u(\boldsymbol{v})\}^{\frac{1}{q}} \left(\underline{\mathbf{u}} - \mathbf{S}_{x_1}^{v_1} \mathbf{S}_{x_2}^{v_2} \mathbb{I}_N \mathbb{U}_N \right) \right\|_q^q \right] \end{aligned} \tag{3.19}$$

where λ is the regularization parameter. We can solve the optimization problem using the steepest descent method [28] (see Appendix C for the gradient term)

$$\widehat{\mathbb{U}}_N^{(\ell+1)} = \widehat{\mathbb{U}}_N^{(\ell)} + \mu \left. \frac{\partial C(\mathbb{U}_N)}{\partial \mathbb{U}_N} \right|_{\mathbb{U}_N = \widehat{\mathbb{U}}_N^{(\ell)}} \tag{3.20}$$

where μ is the step size.

There are two points worth highlighting about the regularization cost function $C_{\mathrm{R}}(\cdot)$ (3.14). First, the regularization term is general enough to subsume several other popular regularization terms existing in the literature, for example, Tikhonov (3.4), TV (3.5), and BTV (3.6). In particular, note that, if we choose the zeroth regression order (i.e. $N = 0$), the proposed regularization term (3.14) becomes

$$C_{\mathrm{R}}(\mathbb{U}_{N=0}) = \sum_{v_1=-\zeta}^{\zeta} \sum_{v_2=-\zeta}^{\zeta} \left\| \{\mathbf{W}_u(\boldsymbol{v})\}^{\frac{1}{q}} \left(\underline{\mathbf{u}} - \mathbf{S}_{x_1}^{v_1} \mathbf{S}_{x_2}^{v_2} \underline{\mathbf{u}} \right) \right\|_q^q. \tag{3.21}$$

Therefor, for $q = 1$, the regularization term (3.21) can be regarded as digital TV [56] with $\mathbf{W}_u(\boldsymbol{v}) = \mathbf{I}$ and $\zeta = 1$. Alternatively, again with $N = 0$, and with $\mathbf{W}_u(\boldsymbol{v}) = \eta^{\|\boldsymbol{v}\|_1}\mathbf{I}$, where $0 \leq \eta \leq 1$, the term represents the BTV regularization criterion first introduced in [2]. That is to say, the BTV implicitly defines the kernel function as

$$K_\eta(\mathbf{x}_i - \mathbf{x}) = \eta^{\|\mathbf{x}_i - \mathbf{x}\|_1}, \tag{3.22}$$

where η in this case is the global smoothing parameter. Second, with the choice of the zeroth order ($N = 0$), we would be assuming that the unknown image $\mathbf{u}$ is piecewise constant. On the other hand, the unknown image is assumed piecewise linear and quadratic with $N = 1$ and 2, respectively. In addition, with the locally data-adaptive weights (introduced in Section 3.2.2), the proposed framework can effectively remove noise and ringing effects while preserving local edges and textures. Of course, other choices of the kernel function are also possible. In the image restoration literature, the use of data-adaptive kernel functions has recently become very popular. As such, one may opt to use mean-shift [35, 36], or nonlocal means [57] weights as the kernel function in the weight matrices $\mathbf{W}_u(\boldsymbol{v})$ and $\mathbf{W}_z(\boldsymbol{v})$ in (3.14) and (3.18), respectively. These methods have implicitly used zeroth-order local representations ($N = 0$), which can also be generalized [58].

3.2.2 Locally Adaptive, Data-Dependant Weights

One fundamental improvement on our kernel-based approach (3.19) can be realized by noting that the kernel regression estimates using the non-adaptive kernel function such as (1.25) and (3.22), independent of the regression order (N), are always local *linear* combinations on the data as described in Section 1.3.3. This is of course relevant to the kernel-based deblurring estimate as well.

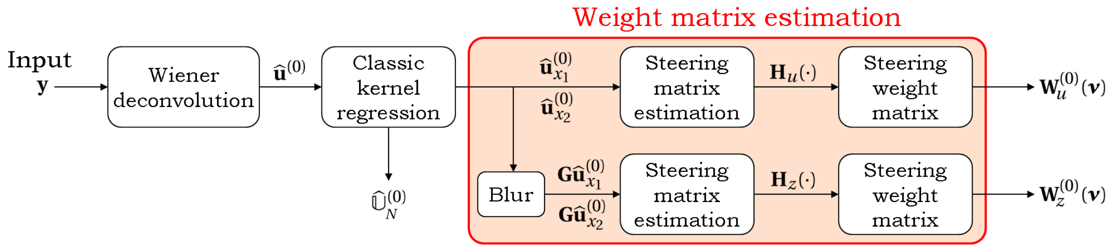
In this section, we describe how to apply the data-adaptive kernel, i.e. the steering kernel function (2.9) for the kernel-based deblurring estimator. The resulting deblurring method can effectively suppress both the noise and ringing effects, which are fundamental issues in the deblurring problem. As we will see later in Section 3.2.3, the choice of the steering kernels is most effective. Furthermore, we will also explain how to implement the data-adaptive kernel-based deblurring method, in which we estimate the unknown image **u** by iteratively updating the image and the data-adaptive weight matrices.

In the kernel-based deblurring approach (3.19), there are two weight matrices $\mathbf{W}_u(\boldsymbol{v})$ for the regularization term and $\mathbf{W}_z(\boldsymbol{v})$ for the likelihood term. Plugging the steering kernel function (2.9) into $\mathbf{W}_u(\boldsymbol{v})$ and $\mathbf{W}_z(\boldsymbol{v})$, we have the data-adapted weight matrices as

$$\mathbf{W}_u(\boldsymbol{v}) = \operatorname{diag}\left\{\cdots,\ K_{\mathbf{H}_u(\mathbf{x}_j-\boldsymbol{v})}(\boldsymbol{v}),\ \cdots\right\} \tag{3.23}$$

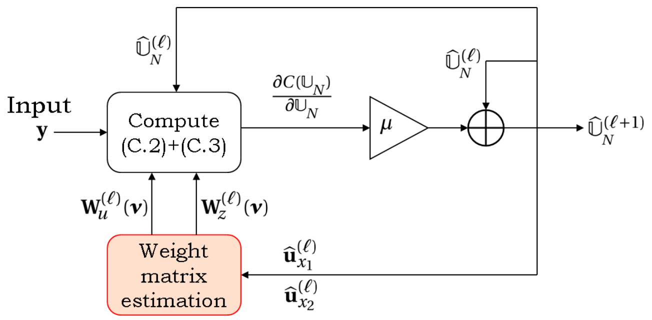
$$\mathbf{W}_z(\boldsymbol{v}) = \operatorname{diag}\left\{\cdots,\ K_{\mathbf{H}_z(\mathbf{x}_j-\boldsymbol{v})}(\boldsymbol{v}),\ \cdots\right\}, \tag{3.24}$$

where $H_u(\cdot)$ and $H_z(\cdot)$ are the steering matrices estimated from the unknown function $u(\cdot)$ of interest and its blurred version $z(\cdot)$, respectively. Figure **3.4** shows a block diagram representation of the kernel-based deblurring method (3.20) with the steering weight matrices (3.23) and (3.24). We note that, using the steering kernel (2.9), the filtering performance depends on the quality of the orientation information. In Section 2.1.3, we presented an algorithm to iteratively refine the orientation information, and then recalculate the steering matrices $\mathbf{H}_i$ by the new orientation estimates. Similarly, for deblurring, we propose an iterative method to refine the steering weight matrices using the steepest descent method 3.20. To begin, we first initialize $\widehat{\mathbb{U}}_N$ and



Weight matrix estimation
Input
$\mathbf{y}$
Wiener deconvolution
$\hat{\mathbf{u}}^{(0)}$
Classic kernel regression
$\hat{\mathbb{U}}_N^{(0)}$
$\hat{\mathbf{u}}_{x_1}^{(0)}$
$\hat{\mathbf{u}}_{x_2}^{(0)}$
Steering matrix estimation
$\mathbf{H}_u(\cdot)$
Steering weight matrix
$\mathbf{W}_u^{(0)}(\boldsymbol{\nu})$
Blur
$\mathbf{G}\hat{\mathbf{u}}_{x_1}^{(0)}$
$\mathbf{G}\hat{\mathbf{u}}_{x_2}^{(0)}$
Steering matrix estimation
$\mathbf{H}_z(\cdot)$
Steering weight matrix
$\mathbf{W}_z^{(0)}(\boldsymbol{\nu})$

(a) Initialization



$\hat{\mathbb{U}}_N^{(\ell)}$
Input
$\mathbf{y}$
Compute (C.2)+(C.3)
$\frac{\partial C(\mathbb{U}_N)}{\partial \mathbb{U}_N}$
μ
$\hat{\mathbb{U}}_N^{(\ell)}$
$\hat{\mathbb{U}}_N^{(\ell+1)}$
$\mathbf{W}_u^{(\ell)}(\boldsymbol{\nu})$
$\mathbf{W}_z^{(\ell)}(\boldsymbol{\nu})$
Weight matrix estimation
$\hat{\mathbf{u}}_{x_1}^{(\ell)}$
$\hat{\mathbf{u}}_{x_2}^{(\ell)}$

(b) Steepest descent iteration with the update of the weight matrix

Figure 3.4: Block diagram representation of the kernel-based deblurring method with the steering weight matrices: (a) initialization and (b) Steepest descent iteration with the update of the weight matrix.

the weight matrices $\mathbf{W}_u(\boldsymbol{\nu})$ and $\mathbf{W}_z(\boldsymbol{\nu})$[1]. To initialize them, we deblur the given image $\mathbf{y}$ by Wiener filter [29] to have a reasonable estimate of the unknown image $\hat{\mathbf{u}}^{(0)}$. Next, by applying the second order ($N = 2$) *classic* kernel regression (1.32), we obtain the first and second derivatives ($\hat{\mathbf{u}}_{x_1}^{(0)}, \hat{\mathbf{u}}_{x_2}^{(0)}, \hat{\mathbf{u}}_{x_1^2}^{(0)}, \hat{\mathbf{u}}_{x_1 x_2}^{(0)}, \hat{\mathbf{u}}_{x_2^2}^{(0)}$), and put them together into $\hat{\mathbb{U}}_2^{(0)}$ as defined in (3.13)[2]. Then, using the first derivatives ($\hat{\mathbf{u}}_{x_1}^{(0)}$ and $\hat{\mathbf{u}}_{x_2}^{(0)}$), we estimate the rotation, elongation, and scaling parameters, θ_i (2.16), σ_i (2.17), and γ_i (2.18), respectively, and create the steering matrices $\mathbf{H}_u(\cdot)$ for every pixel of $\hat{\mathbf{u}}^{(0)}$. Finally, we create the weight matrix $\mathbf{W}_u^{(0)}(v)$ in (3.23) for the regularization term by plugging the steering kernel func-

[1]Although we can start with an image whose are all zero or the given blurry image, a good initialization greatly reduce the number of iterations.

[2]We eliminate the second derivatives for the first regression order $N = 1$.

tion (2.9) with $\mathbf{H}_i = \mathbf{H}_u(\mathbf{x}_i)$. As for the likelihood term, we blur the gradients $\widehat{\mathbf{u}}_{x_1}$ and $\widehat{\mathbf{u}}_{x_2}$ in order to have the gradients of the estimated blurred image. Following the same procedure, with the gradients, we create $\mathbf{W}_z^{(0)}(\boldsymbol{\nu})$. This is the initialization process, and the block diagram is illustrated in Figure **3.4**(a). After the initialization, the iteration process begins with $\widehat{\mathbb{U}}_N^{(0)}$, $\mathbf{W}_u^{(0)}(\nu)$, and $\mathbf{W}_z^{(0)}(\nu)$. Using (C.2) and (C.3) in Appendix C, we compute the gradient term in (3.20), and update $\widehat{\mathbf{U}}_N^{(\ell)}$ and refine the weight matrices ($\mathbf{W}_u^{(\ell)}(\nu)$ and $\mathbf{W}_z^{(\ell)}(\nu)$) with the current estimates of the gradients ($\widehat{\mathbf{u}}_{x_1}^{(\ell)}$ and $\widehat{\mathbf{u}}_{x_1}^{(\ell)}$) contained in $\widehat{\mathbf{U}}_N^{(\ell)}$ alternately in each iterations[3]. Figure **3.4**(b) illustrates the block diagram of the iteration process. As the iteration process continues, the estimated image and its derivatives (and therefore, the orientation information) are gradually improved.

It may be desirable to mention the relationship between the iterative SKR described in Section 2.1.3 and the kernel-based approach (3.20). The iterative SKR smooths the previous estimate without referring to the given noisy data $\mathbf{y}$ during the iteration process. On the other hand, the kernel-based approach (3.20) with the update of the weight matrices keeps the given data in the likelihood term (3.18) and refers to $\mathbf{y}$ at every iteration with a choice of the regularization parameter λ. Neglecting the blurring operation, for a large λ, the kernel-based approach is close to the iterative SKR filtering, whereas for $\lambda = 0$, the kernel-based approach is equivalent to iteratively applying SKR to the given noisy data $\mathbf{y}$ while updating weight matrices by the current estimate.

[3]Alternatively, one can consider updating the weight matrices every few iterations.

3.2.3 Deblurring Examples

In this section, we examine the performance of the kernel-based deblurring method, hereinafter called AKTV[4], while comparing to some existing deblurring algorithms in the literature, we demonstrate the competitiveness of our proposed approach with the existing methods.

First, we replicated an experiment from a two-step deblurring/denoising technique [53]. In this experiment, we restored an image which is degraded by application of a severe blur (19×19 uniform PSF) and addition of white Gaussian noise (BSNR $= 40$[dB][5]). Figure **3.5**(a)-(b) shows the original Cameraman image and the degraded image, and Figure **3.5**(c)-(g) shows the restored images by the Wiener filter [29], a multi-step filter (first denoised by iterative SKR introduced in Section 2.1.3, and then deblurred by the regularization technique (3.3 with BTV (3.6)), ForWaRD[6] [59], LPA-ICI[7] [53], and AKTV (the kernel-based deblurring method (3.20) with $q = 2$, $N = 1$, and steering kernels (2.9)), respectively. The corresponding RMSE values[8] are (b)29.67, (c)17.17, (d)17.39, (e)14.37, (f)13.36, and (g)14.03. Figure **3.5**(h)-(l) shows the zoomed images of Figure **3.5**(a),(d)-(g), respectively. The smoothing parameters for the Wiener filter, the multi-step filter, LPA-ICI, and AKTV are manually optimized to produce the best RMSE values for each method. The default setup for choosing the parameters was used for the ForWaRD method as suggested by the authors. We see that, in this particular (almost noiseless) experiment, the LPA-ICI method results in an image with

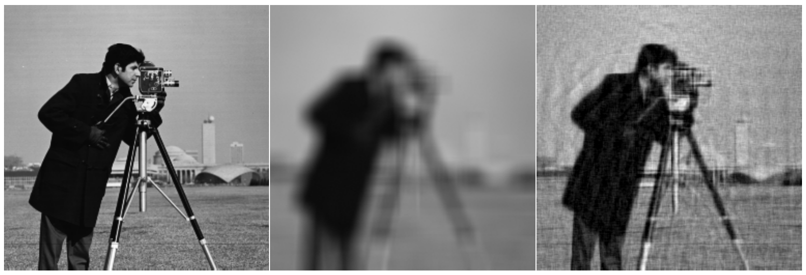
[4]A MATLAB software package containing all the code used to derive the resulted reported in this section is available at `http://www.soe.ucsc.edu/~htakeda/AKTV`.

[5]Blurred signal to noise ratio $= 10\log_{10}(\text{blurred signal vairnace/noise variance})$[dB].

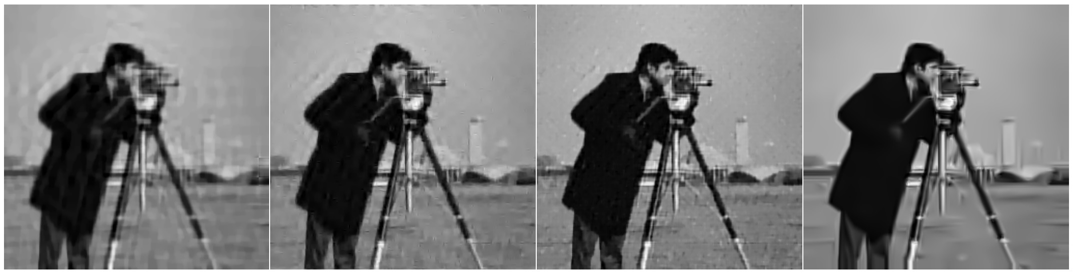
[6]The software is available at `http://www.dsp.rice.edu/software/ward.shtml`.

[7]The software is available at `http://www.cs.tut.fi/~lasip/`.

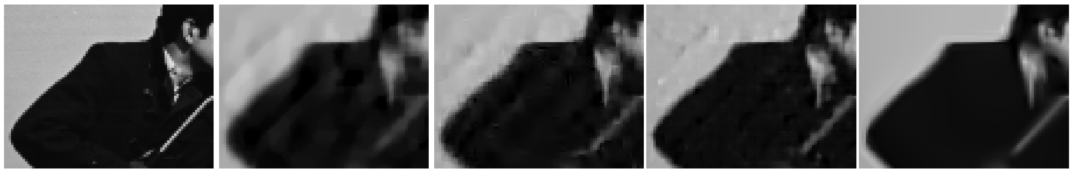
[8]Root mean square error $= \|\text{True image} - \text{The estimated image}\|_2 = \|\underline{\mathbf{u}} - \underline{\widehat{\mathbf{u}}}\|_2$.



(a) Cameraman (256 × 256) (b) Blurry and noisy image RMSE = 29.69 (c) Wiener filter [29] RMSE = 17.17



(d) Multi-step [52, 2] RMSE = 17.39 (e) ForWaRD [59] RMSE = 14.37 (f) LPA-ICI [53] RMSE = 13.36 (g) AKTV (3.20) RMSE = 14.03



(h) The original (i) Multi-step [52, 2] (j) ForWaRD [59] (k) LPA-ICI [53] (l) AKTV (3.20)

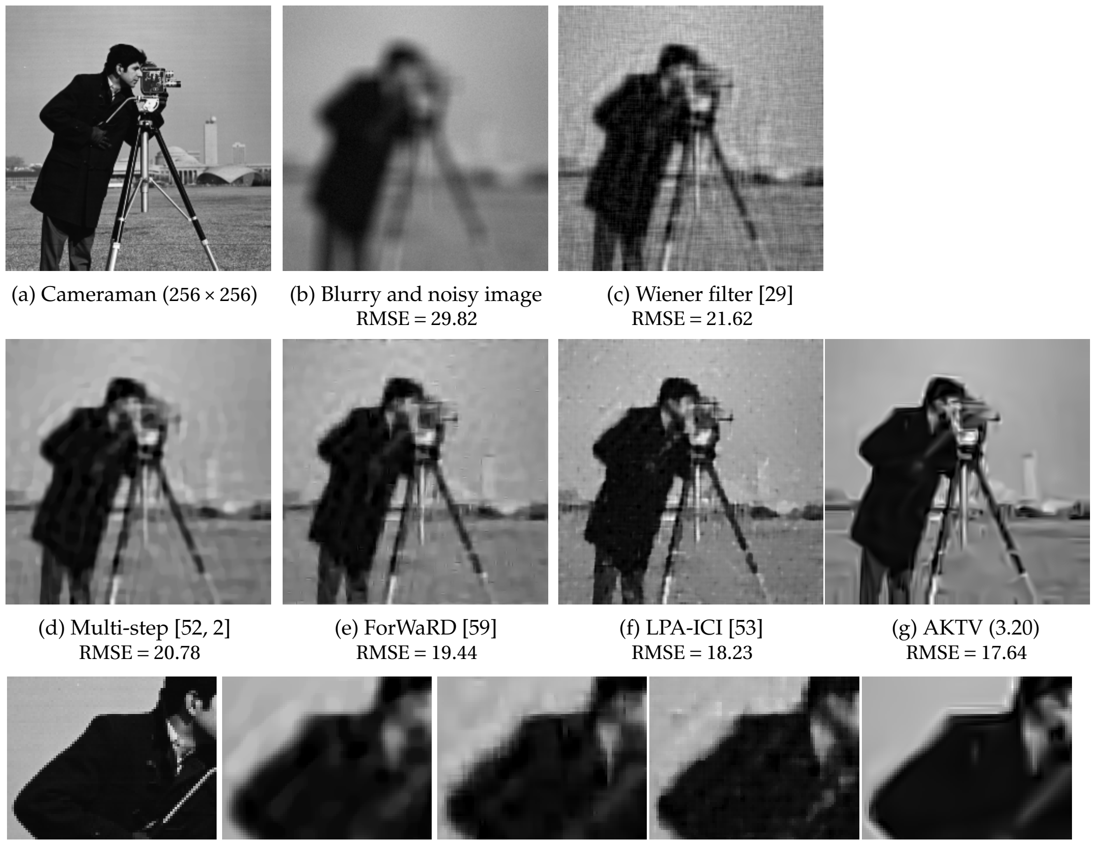
Figure 3.5: Single-frame deblurring experiment with the Cameraman image: (a) the original cameraman image, (b) degraded image by blurring with a 19×19 uniform PSF and adding white Gaussian noise (BSNR = 40[dB]), (c) restored image by Wiener filtering (smoothing parameter 0.0003) [29], (d) restored image by a multi-step filter (first denoised by iterative steering kernel regression [52], and deblurred by BTV [2]), (e) restored image by ForWaRD [59], (f) restored image by LPA-ICI [53], and (g) restored image by AKTV ((3.20) with $q = 1$, $N = 1$, and steering kernels (2.9)). The corresponding RMSE values for (b)-(g) are 29.67, 17.17, 17.39, 14.37, 13.36, and 14.03, respectively. A selected sections of (a) and (d)-(g) are zoomed in (h)-(l), respectively.

slightly better RMSE value than AKTV.

Next, we modified the previous experiment by blurring the Cameraman image by the same PSF (19×19 uniform PSF) and adding much more white Gaussian noise to the blurred image resulting in a degraded image of BSNR = 25[dB]. The resulting image is shown in Figure **3.6**(b) with the RMSE value of 29.82. Similar to the previous experiment, we restored the degraded cameraman image by the Wiener filter, the multi-step filter [52, 2], ForWaRD [59], LPA-ICI [53], and AKTV (the kernel-based deblurring method (3.20) with $q = 2$, $N = 1$, and steering kernels (2.9)). The restored images are shown in Figure **3.6**(c)-(g), respectively, with the RMSE values of (c)21.62, (d)20.78, (e)19.44, (f)18.23, and (g)17.64. Figure **3.6**(h)-(l) shows the zoomed images of Figure **3.6**(a),(d)-(g), respectively. Again, we manually tuned the parameters of each method independently to find the restored image with the best RMSE value. In this noisier example, AKTV resulted in an image with the best RMSE value.

The third deblurring experiment is the case of moderate blur and high noise level. We blur the Lena image with a 5×5 Gaussian PSF with standard deviation (STD) 1.5 and add white Gaussian noise ($\text{BSNR} = 15[\text{dB}]$). The original Lena image and the degraded version are shown in Figure **3.7**(a) and (b), respectively. Similar to the previous experiments, we have compared the reconstruction performance of the Wiener filter [29], ForWaRD [59], LPA-ICI [53], and AKTV ((3.20) with $q = 1$, $N = 1$, and steering kernels (2.9)) in Figure **3.7**(c)-(f). The corresponding RMSE values are (b)10.78, (c)11.18, (d)7.55, (e)6.76, and (f)6.12. Figure **3.7**(g)-(j) shows the zoomed images of Figure **3.7**(a),(d)-(f), respectively.

The last deblurring example addresses the case of a fair amount of blur and noise level. Using the Chemical Plant image which contains fine details shown in Figure **3.8**(a), we generate a blurred image by applying an 11×11 Gaussian PSF with STD 1.75 and adding white Gaussian noise ($\text{BSNR} = 30[\text{dB}]$), and the degraded image is shown in Figure **3.8**(b). Again, similar to the previous experiments, the restored images by the Wiener filter [29], ForWaRD [59], LPA-ICI [53], and AKTV ((3.20) with $q = 1$,

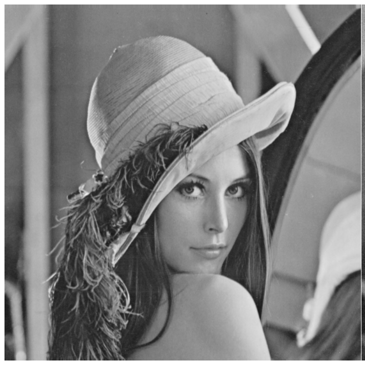

(a) Cameraman (256 × 256) (b) Blurry and noisy image RMSE = 29.82 (c) Wiener filter [29] RMSE = 21.62

(d) Multi-step [52, 2] RMSE = 20.78 (e) ForWaRD [59] RMSE = 19.44 (f) LPA-ICI [53] RMSE = 18.23 (g) AKTV (3.20) RMSE = 17.64

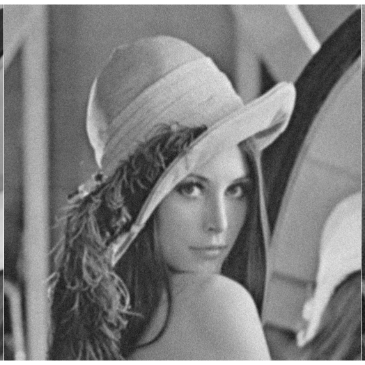
(h) The original (i) Multi-step [52, 2] (j) ForWaRD [59] (k) LPA-ICI [53] (l) AKTV (3.20)

Figure 3.6: Single-frame deblurring experiment on the Cameraman image: (a) original cameraman image, (b) degraded image by blurring with a 19×19 uniform PSF and adding white Gaussian noise (BSNR = 25[dB]), (c) restored image by Wiener filtering (the smoothing parameter 0.004 [29]), (d) restored image by a multi-step filter (first denoised by iterative steering kernel regression [52], and deblurred by BTV [2]) (e) restored image by ForWaRD [59], (f) restored image by LPA-ICI [53], and (g) restored image by AKTV ((3.20) with $q = 1$, $N = 1$, and steering kernels (2.9)). The corresponding RMSE values for (b)-(g) are 29.82, 21.62, 20.78, 19.44, 18.23, and 17.64, respectively. A selected sections of (a) and (d)-(g) are zoomed in (h)-(l), respectively.

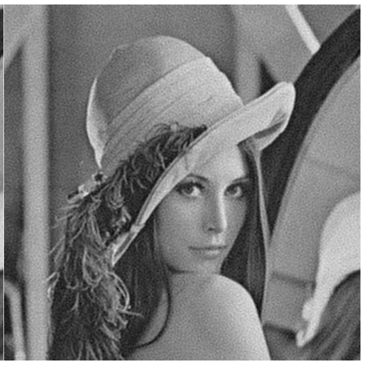
$N = 1$, and steering kernels (2.9)) are shown Figure **3.8**(c)-(f), respectively. The corresponding RMSE values are (b)15.09, (c)9.29, (d)8.98, (e)8.98, and (f)8.57. In the last two experiments, the restored images by AKTV again show the best numerical and visual performance.



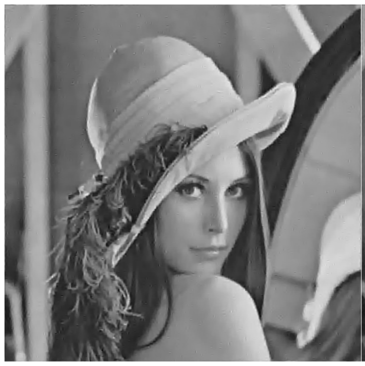
(a) Lena (512×512)



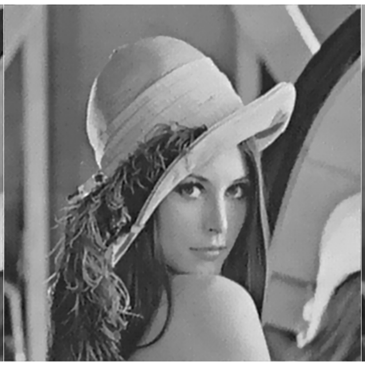
(b) Blurry and noisy image
RMSE = 10.78



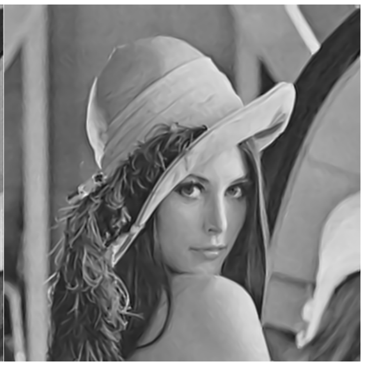
(c) Wiener filter [29]
RMSE = 11.18



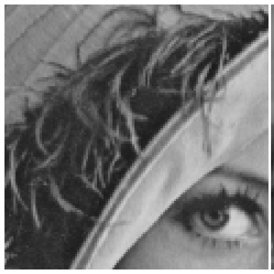
(d) ForWaRD [59]
RMSE = 7.55



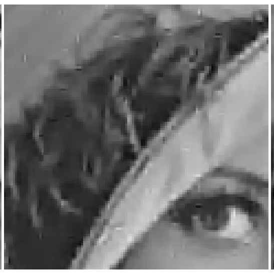
(e) LPA-ICI [53]
RMSE = 6.76



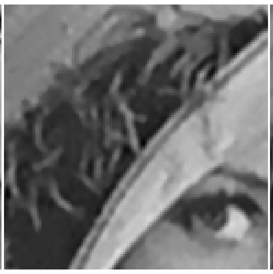
(f) AKTV (3.20)
RMSE = 6.12



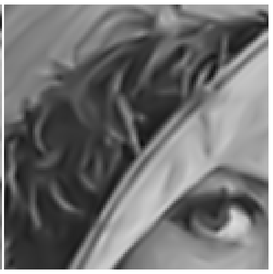
(g) The original



(h) ForWaRD [59]

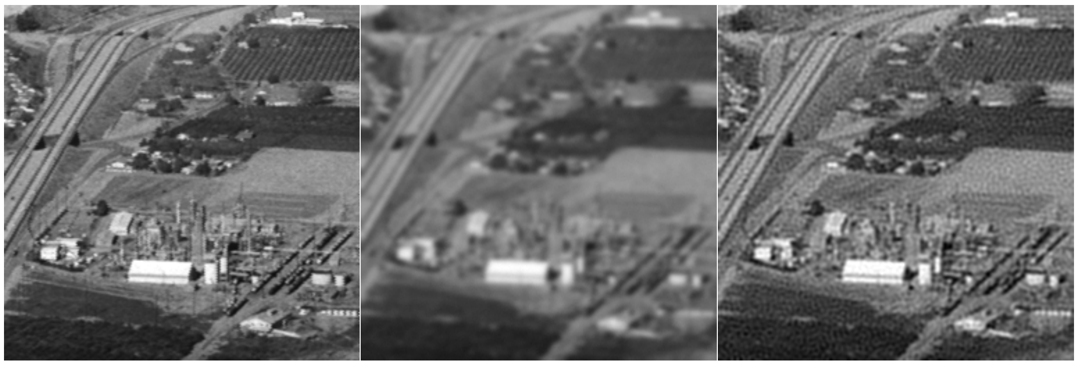


(i) LPA-ICI [53]



(j) AKTV (3.20)

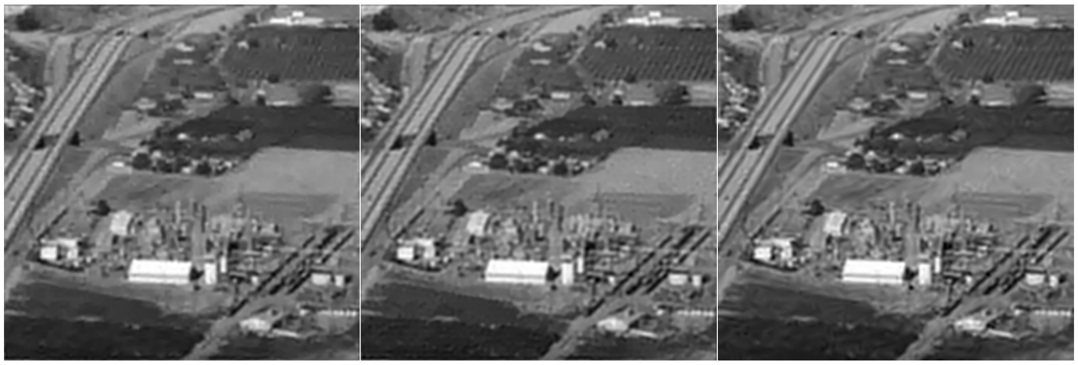
Figure 3.7: Single-frame deblurring simulation of the Lena image: (a) original Lena image, (b) degraded image by blurring with a 5×5 Gaussian PSF (STD = 1.5) and adding white Gaussian noise (BSNR = 15[dB]), (c) restored image by the Wiener filter method with smoothing parameter of 0.05 [29], (d) restored image by ForWaRD [59], (e) restored image by LPA-ICI [53], and (f) restored image by AKTV ((3.20) with $q = 1$, $N = 1$, and steering kernels (2.9)). The corresponding RMSE values for (b)-(f) are 10.78, 11.18, 7.55, 6.76, and 6.12, respectively. A selected sections of (a) and (d)-(f) are zoomed in (g)-(j), respectively.



(a) Chemical plant (256×256)

(b) Blurry and noisy image
RMSE = 15.09

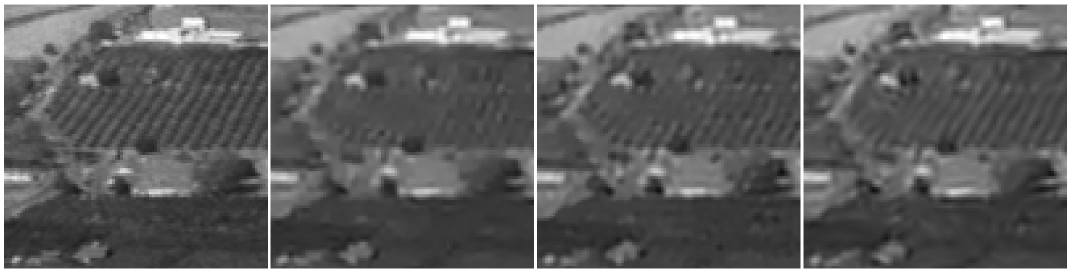
(c) Wiener filter [29]
RMSE = 9.29



(d) ForWaRD [59]
RMSE = 8.98

(e) LPA-ICI [53]
RMSE = 8.98

(f) AKTV (3.20)
RMSE = 8.57



(g) The original

(h) ForWaRD [59]

(i) LPA-ICI [53]

(j) AKTV (3.20)

Figure 3.8: Single-frame deblurring simulation of the Chemical Plant image: (a) original Lena image, (b) degraded image by blurring with a 11×11 Gaussian PSF (STD = 1.75) and adding white Gaussian noise (BSNR = 30[dB]), (c) restored image by the Wiener filter method with smoothing parameter of 0.01 [29], (d) restored image by ForWaRD [59], (e) restored image by LPA-ICI [53], and (f) restored image by AKTV ((3.20) with $q = 1$, $N = 1$, and steering kernels (2.9)). The corresponding RMSE values for (b)-(f) are 15.09, 9.29, 8.98, 8.98, and 8.57, respectively. A selected sections of (a) and (d)-(f) are zoomed in (g)-(j), respectively.

3.3 Generalized Kernel-Based Image Restoration

Considering the effect of general distortions or transformation, beyond the blurring operation $\mathbf{H}$ in the data model (3.2), the kernel-based approach can be again applicable to denoising and interpolation. Denoting the general distortion operation by $\mathbf{\Phi}$, we rewrite the data model (3.2) as

$$\underline{\mathbf{y}} = \underline{\mathbf{z}} + \underline{\boldsymbol{\varepsilon}} = \mathbf{\Phi}\underline{\mathbf{u}} + \underline{\boldsymbol{\varepsilon}}, \tag{3.25}$$

where the dimension of $\mathbf{\Phi}$ is $\mathscr{R}^{L'M' \times LM}$, $\mathbf{y} \in \mathscr{R}^{L' \times M'}$ is the measured signal, $\mathbf{z} \in \mathscr{R}^{L' \times M'}$ is the noise-free distorted signal, $\boldsymbol{\varepsilon} \in \mathscr{R}^{L' \times M'}$ is the noise signal, and $\mathbf{u} \in \mathscr{R}^{L \times M}$ is the unknown image or interest. The dimensions of $\mathbf{y}$ and $\mathbf{u}$ may be different due to the distortion operation, e.g. the downsampling operation.

Now, we modify the cost function of the kernel-based approach taking the new data model (3.25) into account:

$$\begin{aligned} C(\mathbb{U}_N) &= C_{\mathrm{L}}(\mathbb{U}_N) + \lambda C_{\mathrm{R}}(\mathbb{U}_N) \\ &= \sum_{\nu_1=-\zeta}^{\zeta} \sum_{\nu_2=-\zeta}^{\zeta} \left[\left\| \{\mathbf{W}_z(\boldsymbol{\nu})\}^{\frac{1}{2}} \left(\underline{\mathbf{y}} - \mathbf{\Phi} \mathbf{S}_{x_1}^{\nu_1} \mathbf{S}_{x_2}^{\nu_2} \mathbb{I}_N \mathbb{U}_N \right) \right\|_2^2 + \lambda \left\| \{\mathbf{W}_u(\boldsymbol{\nu})\}^{\frac{1}{q}} \left(\underline{\mathbf{u}} - \mathbf{S}_{x_1}^{\nu_1} \mathbf{S}_{x_2}^{\nu_2} \mathbb{I}_N \mathbb{U}_N \right) \right\|_q^q \right]. \end{aligned} \tag{3.26}$$

Using the generalized kernel-based approach (3.19), we conducted two experiments: a denoising example and a simultaneous denoising-interpolation example. The first denoising example is shown in Figure **3.9**, where the operator $\mathbf{\Phi}$ is the identity matrix, i.e. $\mathbf{\Phi} = \mathbf{I} \in \mathscr{R}^{LM \times LM}$, and we used a real high resolution X-ray image of a chicken wing[9] obtained by a super-resolution X-ray imaging method [60] shown in Figure **3.9**(b). By contrast, Figure **3.9**(a) shows a traditional X-ray of the same object, which is less

[9] The image is available at `http://blogs.zdnet.com/emergingtech/?p983`.

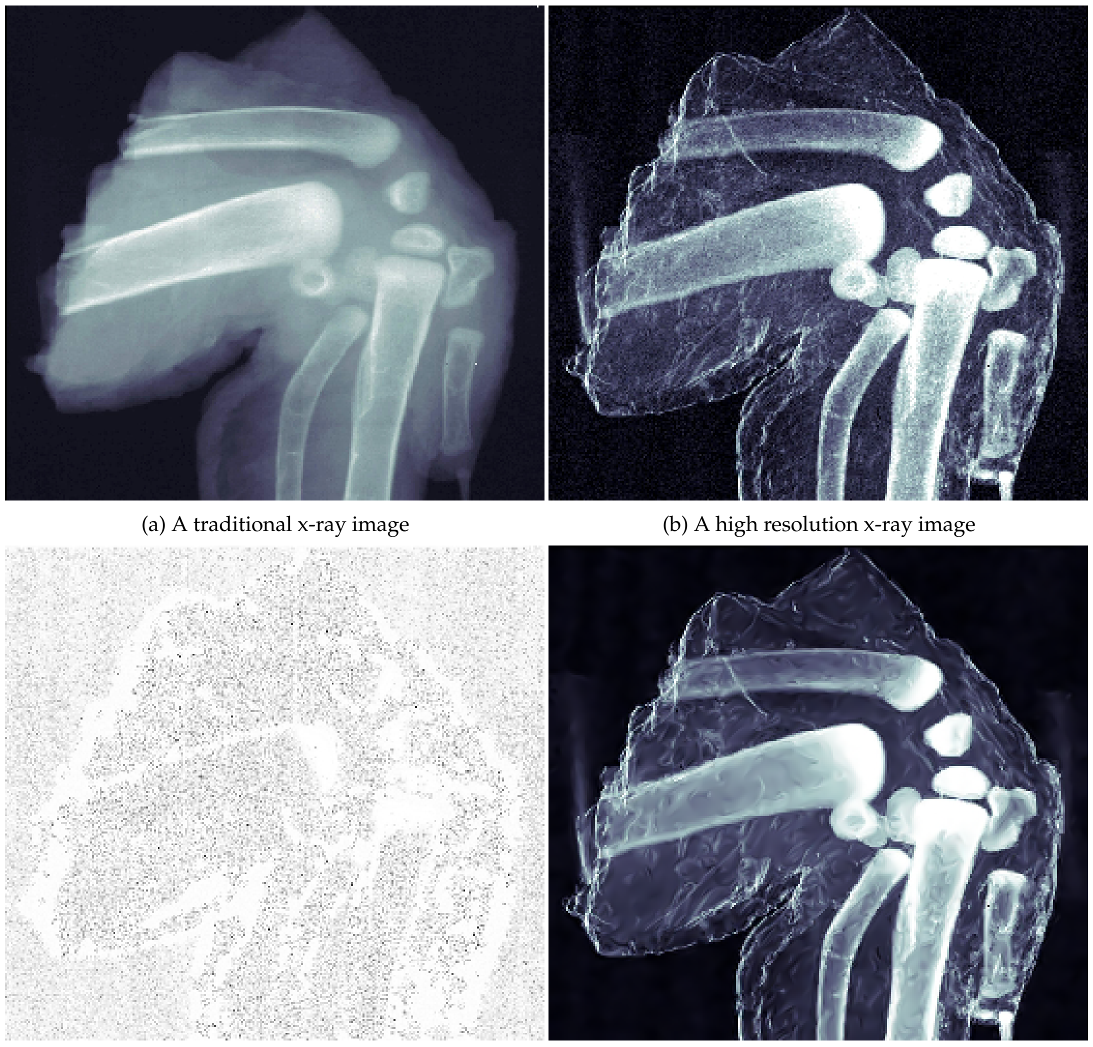
noisy, but also considerably less detailed. Therefore, effective denoising of the high-resolution X-ray image can provide unprecedented levels of detail and clarity for this imaging modality. The denoised result by AKTV ((3.19) with $\mathbf{\Phi} = \mathbf{I} \in \mathscr{R}^{LM \times LM}$, $q = 1$, $N = 1$, and steering kernels (2.9)) is shown in Figure **3.9**(c). Figure **3.9**(d) illustrates the absolute residual image (the absolute difference between the given noisy image (Figure **3.9**(b)) and the denoised image), which is intended to show that AKTV effectively removed unknown noise while preserving structures.

The second example is image upscaling. For upscaling, the operator $\mathbf{\Phi}$ is a downsampling operator, i.e. $\mathbf{\Phi} = \mathbf{D} \in \mathscr{R}^{L'M' \times LM}$, where $L' = \frac{L}{r}$, $M' = \frac{M}{r}$, and r is the downsampling factor. Figure **3.10**(a) shows a real low-resolution MRI image (256×256) of a human head, which we obtained from the Whole Brain Atlas[10]. The upscaled image with $r = 2$ by bicubic interpolation[11], NEDI [61], and AKTV ((3.19) with $\mathbf{\Phi} = \mathbf{D} \in \mathscr{R}^{\frac{L}{2}\frac{M}{2} \times LM}$, $q = 1$, $N = 1$, and steering kernels (2.9)) are shown in Figures **3.10**(b)-(d), respectively. The example shows that the proposed method, AKTV, is capable of upscaling and denoising an image simultaneously, and such a one-step approach is generally more suitable than a multiple-step approach (e.g., denoising followed by upscaling).

In this section, we illustrated the applicability of an advanced nonparametric approach to the treatment of images from a variety of modalities. The approach is sufficiently general so that, as shown in Figures **3.9** and **3.10**, it is an effective tool for denoising and interpolation as well as deblurring. It is of particular note that the approach makes minimal assumptions about the global structure of the signal and noise, and is therefore, quite generally useful. It is also worth stating that the distortion op-

[10]`http://www.med.harvard.edu/AANLIB/cases/caseNA/gr/cor/051.png`

[11]We used a MATLAB routine "`interp2`" with `cubic` option.


(a) A traditional x-ray image

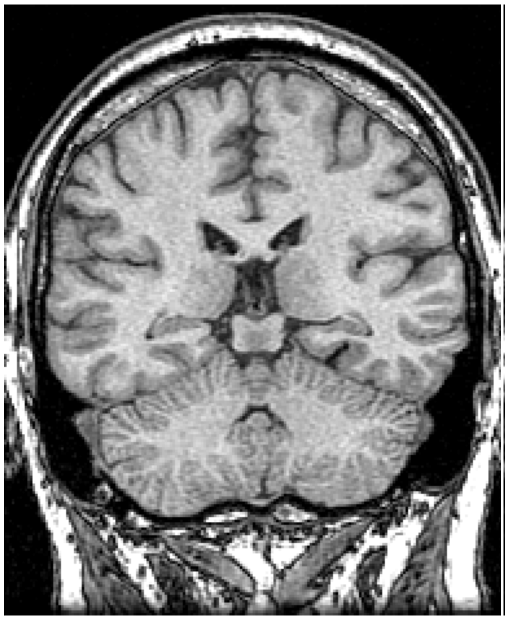
(b) A high resolution x-ray image

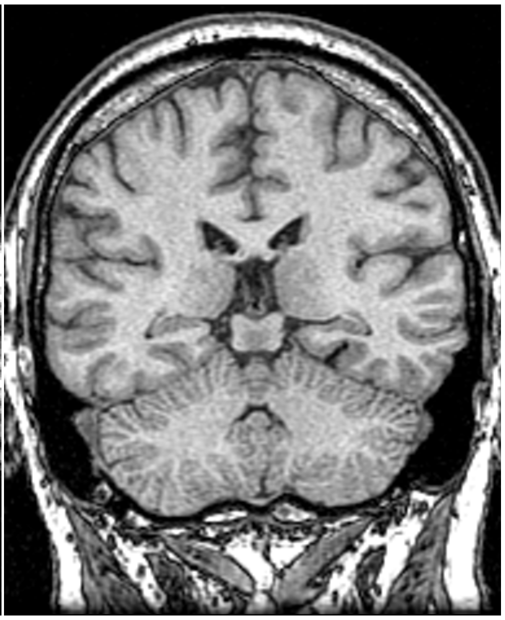
(d)Absolute residual image between (b)and(c)

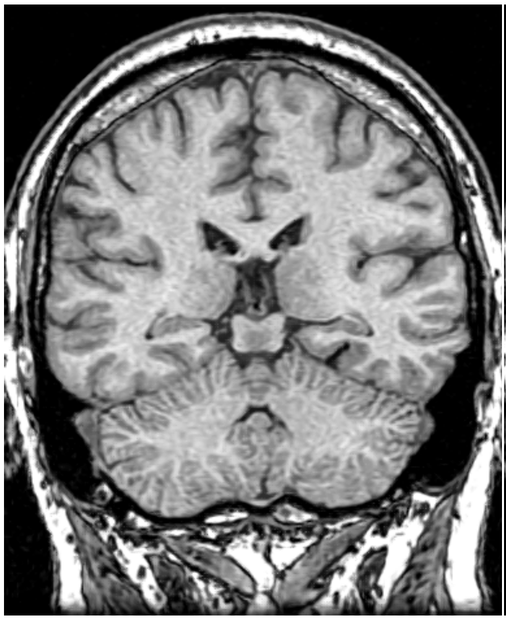
(c) AKTV, applied to (b)

Figure 3.9: A denoising example by AKTV: (a) a traditional x-ray image of a chicken wing, (b) a high resolution x-ray image, (c) the denoised image by AKTV ((3.19) with $\mathbf{\Phi} = \mathbf{I} \in \mathscr{R}^{LM \times LM}$, $q = 1$, $N = 1$, and steering kernels (2.9)), and (d) absolute residual image.

erator $\mathbf{\Phi}$ can be, for instance, a combination of the downsampling and blur operations. Furthermore, the kernel-based approach is also applicable to the super resolution problem [2] where multiple images are fused to provide a higher resolution image.


(a) A low resolution MRI image


(b) ×2 Upscaling by bicubic interpolation


(c) ×2 Upscaling by NEDI [61]

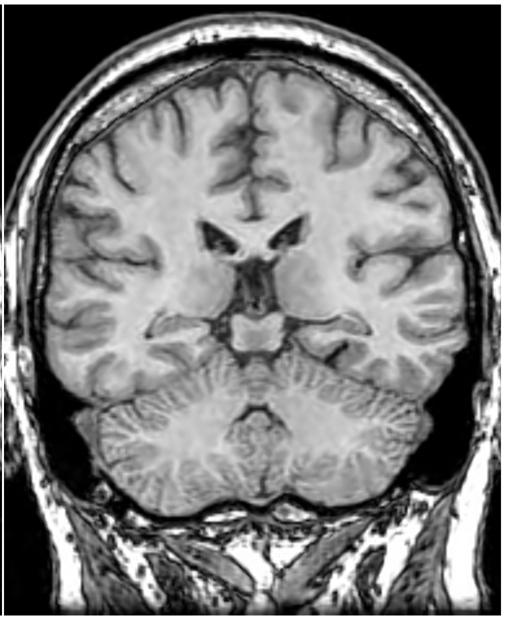

(d) ×2 Upscaling by AKTV

Figure 3.10: An upscaling example: (a) a low resolution MRI image of a human head, and (b)-(d) ×2 upscaled images by bicubic, NEDI [61], and AKTV ((3.19) with $\mathbf{\Phi} = \mathbf{D} \in \mathscr{R}^{\frac{L}{r}\frac{M}{r} \times LM}$, the upscaling factor $r = 2$, $q = 1$, $N = 1$, and steering kernels (2.9)), respectively.

Summary— In this chapter, we extended the data-adaptive kernel regression framework introduced in Chapter 2, which was earlier used for denoising and interpolation, for the deblurring application. Numerical and visual comparison with the existing deblurring techniques showed the effectiveness of the proposed technique in the low-SNR cases, while in the high-SNR cases the performance was comparable to the best technique in the literature. In addition, the kernel-based deblurring approach produces less artifacts including ringing artifacts compared to the Wiener filter, ForWaRD, and LPA-ICI.

Also, the proposed image prior (3.14) in its general form subsumes some of the most popular image priors in the literature (Tikhonov (3.4), digital TV (3.5), and BTV (3.6)). It is of course possible to create other image prior based on it by choosing local representation other than Taylor series or using other type of kernel functions, e.g., bilateral kernel [7], non-local means [57]. Finally, we described the generalized form of the kernel-based approach and showed its applicability to a wide variety of image processing problems.

Chapter 4

Multi-Dimensional Kernel Regression for Space-Time Video Upscaling

Abstract— In this chapter, we develop an adaptive enhancement method for video-to-video application including not only denoising but also spatiotemporal upscaling of videos which possibly contain complex motions. Our approach is based on *multidimensional kernel regression*, where each pixel in the video sequence is now approximated with a 3-D local representation (i.e. 3-D Taylor series), capturing the essential local behavior of its spatiotemporal neighborhood. The coefficients of the series are estimated by solving a local weighted least-squares problem, where the weights are given by a 3-D (spatiotemporal) kernel function taking both local spatial orientations and local motion trajectories in the neighborhood into account. This chapter presents two approaches to construct the 3-D local kernels: (i) we separately estimate spatial orientations and local motions, and then explicitly embed them in the local kernels to compute adaptive weights for neighborhoods, and (ii) a 3-D version of steering kernel regression where we estimate spatial orientations and local motions together

implicitly 3-D (spatiotemporal) orientations and then compute adaptive weights. We call the former approach *motion assisted steering kernel regression* (MASK) and the latter approach *3-D steering kernel regression* (3-D SKR). In Section 4.2 and Section 4.3, we describe MASK and 3-D SKR in details, respectively.

4.1 Introduction

The emergence of high definition displays in recent years (e.g. 720×1280 and 1080×1920 or higher spatial resolution, and up 240Hz in temporal resolution), along with the proliferation of increasingly cheaper digital imaging technology has resulted in the need for fundamentally new image processing algorithms. Specifically, in order to display relatively low quality content on such high resolution displays, the need for better space-time upscaling, denoising, and deblurring algorithms has become an urgent market priority, with correspondingly interesting challenges for the academic community. The goal for spatial and temporal video interpolation/reconstruction is to enhance the resolution of the input video in a manner that is visually pleasing and artifact-free. Common visual artifacts that may occur in spatial and temporal interpolation are edge jaggedness, ringing, blurring, of edges and texture detail, as well as motion blur and judder. In addition, the input video usually contains noise and other artifacts caused by compression. Due to the increasing size of modern video displays, as well as incorporation of new display technologies (e.g. higher brightness, wider color gamut), artifacts in the input video and those introduced by scaling are amplified, and become more visible than with past display technologies. High quality video upscaling requires resolution enhancement and sharpness enhancement as well as noise and compression artifact reduction.

A common approach for spatial image and video upscaling is to use linear filters with compact support, such as from the family of cubic filters [62]. In this chapter, our focus is on multi-frame methods, which enable resolution enhancement in spatial upscaling, and allow temporal frame interpolation (also known as frame rate upconversion). Although many algorithms have been proposed for image and video interpolation, spatial upscaling and frame interpolation (temporal upscaling) are generally treated separately. The existing literature on enhancement and upscaling (often called super-resolution[1]) is vast and rapidly growing in both the single frame case [63, 64] and the multi frame (video) case [2, 65, 66, 67, 68, 69, 70, 71, 72], and many new algorithms for this problem have been proposed recently. Yet, one of the most fundamental roadblocks have not been overcome. In particular, in order to be effective, essentially all the existing multi-frame super-resolution approaches must perform (sub-pixel) accurate motion estimation [2, 65, 66, 67, 68, 69, 70, 71, 72, 73]. As a result, most methods fail to perform well in the presence of complex motions which are generally present in videos. Indeed, in most practical cases where complex motion and occlusions are present and not estimated with pinpoint accuracy, existing algorithms tend to fail catastrophically, often producing outputs that are of even worse visual quality than the low-resolution inputs.

For temporal upscaling, a motion-based technique called *motion compensated frame interpolation* is popular. In [74], Fujiwara *et al.* extract motion vectors from a compressed video stream for motion compensation. However, these motion vectors are often unreliable; hence they refine the motion vectors by the block matching approach

[1]To clarify the use of words *super-resolution* and *upscaling*, we note that if the algorithm does not receive input frames that are aliased, it will still produce an output with a higher number of pixels and/or frames (i.e. “upscaled”), but which is not necessarily “superresolved”.

with variable-size blocks. Similar to Fujiwara's work, in [75], Huang *et al.* proposed another refinement approach for motion vectors. Using the motion reliability computed from prediction errors of neighboring frames, they smooth the motion vector field by employing a vector median filter with weights decided based on the local motion reliability. In [76, 77], instead of refining the motion vector field, Kang *et al.* and Choi *et al.* proposed block matching motion estimation with overlapped and variable-size block technique in order to estimate motion as accurately as possible. However, the difficulty of the motion-based approach is that, even though the motion vector field may be refined and/or smoothed, more complex transitions (e.g. occlusions, transparency, reflection, and non-rigidity) are not accurately treated. That is, motion errors are inevitable even after smoothing/refining motion vector fields, and, hence, an appropriate mechanism that takes care of the errors is necessary for producing artifact-free outputs.

In this chapter, we will present two different spatiotemporal video upscaling methods based on *multi-dimensional (3-D) kernel regression* with the steering kernel function presented in Chapter 2: *motion-assisted steering kernel* (MASK) *regression*, and *3-D steering kernel regression* (3-D SKR). First, in Section 4.2, we describe the MASK approach, where the MASK function computes the adaptive weights for the neighboring samples. The MASK function is a 3-D function and it explicitly takes spatial (2-D) orientation and the local motion trajectory into account. Then, the MASK function utilizes an analysis of the spatial orientation and the local motion vectors to steer spatiotemporal regression kernels. Subsequently, local kernel regression is applied to weighted least-squares to find optimal pixel estimates. Although 2-D kernel regression has been applied to achieve super-resolution reconstruction through fusion of multiple pre-registered frames on to a 2-D plane in Chapter 2, the MASK approach is different in that it does not require explicit motion compensation of the video frames. Instead, we use 3-D weighting kernels that are "warped" accordingly to estimated motion vectors, such that the regression process acts directly upon the video data. Although we consider local motion vectors in MASK, we propose an algorithm that is robust against errors in the estimated motion field. Prior multi-frame resolution enhancement or super-resolution (SR) reconstruction methods often consider only global translational or affine motions; local motion and object occlusions are often not addressed. Many SR methods [2, 63, 65, 66, 67, 68, 69, 70, 71, 72, 73] require explicit motion compensation, which may involve interpolation or rounding of displacements to grid locations. These issues can have a negative impact on accuracy and robustness. The MASK approach is capable of handling local motions, avoids explicit motion compensation, and is more robust to errors in the estimated motion field.

Next, in Section 4.3, addressing the challenging problem of spatiotemporal video super-resolution in a fundamentally different way, which removes the need for explicit subpixel accuracy motion estimation, we introduce the 3-D SKR approach. We present a methodology that is based on the notion of consistency between the estimated pixels, which is derived from the novel use of kernel regression [52, 78]. Classic kernel regression is a well-suited, non-parametric point estimation procedure. In Chapter 2, we generalized the use of these technique to spatially adaptive (steering) kernel regression, which produces results that preserve and restore details with minimal assumptions on local signal and noise models [3]. Other related non-parametric techniques for multidimensional signal processing have emerged in recent years as well. In particular, the concept of normalized convolution [5], and the introduction of support vector machines [79] are notable examples.

In a related work [80] to the 3-D SKR approach, we have generalized the non-local means (NLM) framework [57] to the problem of super-resolution. In that work, measuring the similarity of image *patches* across space and time resulted in "fuzzy" or probabilistic motions, which is well evaluated in [81] by Protter *et al.*. Such estimates also avoid the need for explicit motion estimation and give relatively larger weights to more similar patches used in the computation of the high resolution estimate. Another example of a related approach appears in [82] where Danielyan *et al.* have presented an extension of the block-matching 3-D filter (BM3D) [45] for video super-resolution, in which explicit motion estimation is also avoided by classifying the image patches using a block matching technique. The objectives of the present work, the NLM-based approach [80], and Video-BM3D [82] just mentioned are the same: namely, to achieve super-resolution on general sequences, while avoiding explicit (subpixel-accurate) motion estimation. These approaches represent a new generation of super-resolution algorithms that are quite distinctly different from all existing super-resolution methods. Specifically, existing methods have required highly accurate subpixel motion estimation and have thus failed to achieve resolution enhancement on arbitrary sequence.

To sum up, both MASK and 3-D SKR encompass super-resolution in 3-D as well as denoising and spatiotemporal upscaling. The framework is based on the development of locally adaptive 3-D filters with coefficients depending on the pixels in a local neighborhood of interest in space-time in a novel way. These filter coefficients are computed using a particular measure of similarity and consistency between the neighboring pixels which uses the local geometric and radiometric structure of the neighborhood.

4.2 Motion-Assisted Steering Kernel (MASK) Regression

SKR estimates an unknown pixel value in a single image by a weighted combination of neighboring pixels that belong to the same image, giving larger weights to the pixels along orientation. In this section, we develop a multi-frame video upscaling method based on SKR by additionally utilizing local motion vectors, and we call the resulting method *motion-assisted steering kernel* (MASK) *regression*. The MASK approach is a 3-D kernel regression method in which the pixel of interest is estimated by a weighted combination of pixels in its spatiotemporal neighborhood involving multiple video frames. Hence, we first extend the 2-D kernel regression framwork into a 3-D framework introducing the time axis. Then we present our 3-D data-adaptive kernel, the MASK function, which relies not only on local spatial orientation but also local motion trajectory. Then, we describe the process of spatial upscaling and temporal frame interpolation based on MASK. While we focus on the principle of our approach in this section, we present a specific algorithm for video processing based on the MASK approach later in this section.

4.2.1 Spatiotemporal (3-D) Kernel Regression

The extension of the 2-D kernel regression into 3-D is straightforward. Introducing the time axis to the data model (1.17), we have the 3-D data model as

$$y_i = z(\mathbf{x}_i) + \varepsilon_i, \quad \mathbf{x}_i \in \omega, \quad i = 1, \cdots, P, \tag{4.1}$$

where y_i is a noise-ridden measurement at $\mathbf{x}_i = [x_{1i}, x_{2i}, t_i]^T$, x_{1i} and x_{2i} are the spatial coordinates, $t_i (= x_{3i})$ is the temporal coordinate, $z(\cdot)$ is the trivariate regression function of interest, ε_i is an *i.i.d.* zero-mean noise process, and P is the total number of nearby samples in a 3-D neighborhood ω of interest, where we will henceforth call ω a "cubicle".

Similar to the 2-D case, in order to estimate the value of $z(\mathbf{x})$ at an arbitrary position $\mathbf{x}$, given the above data samples y_i, we can rely on a local N-th order Taylor expansion about $\mathbf{x}$. We denote the pixel value of interest $z(\mathbf{x})$ by β_0, while $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \cdots, \boldsymbol{\beta}_N$ denote vectors containing first-order, second-order, $\cdots$, N-th order partial derivatives of $z(\mathbf{x})$ at $\mathbf{x}$, resulting from the Taylor expansion, For instance, β_0 and $\boldsymbol{\beta}_1$ are defined as

$$\begin{aligned} \beta_0 &= z(\mathbf{x}), & (4.2) \\ \boldsymbol{\beta}_1 &= \begin{bmatrix} z_{x_1}(\mathbf{x}) & z_{x_2}(\mathbf{x}) & z_t(\mathbf{x}) \end{bmatrix}^T. & (4.3) \end{aligned}$$

Regardless of the dimensionality of $\mathbf{x}$, the unknown parameters $\boldsymbol{\beta}_n$ for $n = 0, \cdots, N$ can be estimated from y_i with $i = 1, \cdots, P$ using the kernel regression optimization procedure:

$$\min_{\{\boldsymbol{\beta}_n\}_{n=0}^{N}} \sum_{i=1}^{P} \left[y_i - \beta_0 - \boldsymbol{\beta}_1^T(\mathbf{x}_i - \mathbf{x}) - \boldsymbol{\beta}_2^T \text{vech}\left\{ (\mathbf{x}_i - \mathbf{x})(\mathbf{x}_i - \mathbf{x})^T \right\} - \cdots \right]^2 K_{\mathbf{H}_i}(\mathbf{x}_i - \mathbf{x}), \tag{4.4}$$

where N is the regression order and $K(\cdot)$ is the kernel function that weights the influence of each nearby samples. $\mathbf{H}_i$ is now a 3×3 smoothing matrix of the i-th sample (y_i), and the rest of this chapter focuses on how we incorporate the local spatial and temporal structures into $\mathbf{H}_i$ in two different ways so that the resulting estimator provides a pixel in high quality. Similar to the 1-D and 2-D cases, the optimization (4.4) yields an estimator which computes a local unknown pixel by a weighted linear combination of nearby samples:

$$\hat{z}(\mathbf{x}) = \widehat{\beta}_0 = \sum_{i=1}^{P} W_i(K, \mathbf{H}_i, N, \mathbf{x}_i - \mathbf{x})\, y_i, \tag{4.5}$$

where $W_i(\cdot)$ is the equivalent kernel weight function in 3-D.

4.2.2 Motion-Assisted Steering Kernel Function

A good choice for the steering matrices spatiotemporally is to consider local motion or optical flow vectors caused by object displacements across frames, in conjunction with spatial steering along local edges and isophotes. Spatial steering should consider the locally dominant orientation of the pixel data and should allow elongation of the kernel in this direction. Spatiotemporal steering should allow alignment of the kernel weights with the local optical flow or motion trajectory, as well as overall temporal scaling. Hence, we construct our spatiotemporal kernel as a product of a spatial- and motion-steering kernel, and a kernel that acts temporally:

$$K^{\mathrm{MASK}}(\mathbf{x}_i - \mathbf{x}) = \frac{1}{\det\left(\mathbf{H}_i^{\mathrm{s}}\right)} K\left\{\left(\mathbf{H}_i^{\mathrm{s}}\right)^{-1} \mathbf{H}_i^{\mathrm{m}}(\mathbf{x}_i - \mathbf{x})\right\} K_{h_{\mathrm{t}}}(t_i - t), \tag{4.6}$$

where $\mathbf{H}_i^{\mathrm{s}} \in \mathscr{R}^{3\times 3}$ is a *spatial steering matrix*, $\mathbf{H}_i^{\mathrm{m}} \in \mathscr{R}^{3\times 3}$ is a *motion steering matrix*, the second kernel $K_{h_{\mathrm{t}}}(\cdot)$ is a temporal kernel function, and h_{t} is the temporal smoothing parameter which controls the temporal penalization. These data-dependent kernel components determine the steering action at the position of the i-th sample $\mathbf{x}_i$, and are described next.

Following Section 2.1.2, we define the spatial steering matrix $\mathbf{H}_i^{\mathrm{s}}$ for the MASK function (4.6) as

$$\mathbf{H}_i^{\mathrm{s}} = h_{\mathrm{s}} \begin{bmatrix} \mathbf{C}_i & \\ & 1 \end{bmatrix}^{-\frac{1}{2}} = h_{\mathrm{s}} \mathbb{C}_i^{-\frac{1}{2}}, \tag{4.7}$$

where h_{s} is a global spatial smoothing parameter, and $\mathbf{C}_i \in \mathscr{R}^{2\times 2}$ is the covariance matrix which is given by (2.13) and captures the sample variations in a local spatial neighborhood around $\mathbf{x}_i$. All the covariance matrices $\mathbf{C}_i$'s are constructed in a parametric manner as described in (2.13).

Next, the motion steering matrix $\mathbf{H}_i^{\mathrm{m}}$ is constructed on the basis of a local estimate of the motion (or optical flow vector) $\mathbf{m}_i = [m_{1i}, m_{2i}]^T$ at $\mathbf{x}_i$. Namely, we warp the kernel along the local motion trajectory using the following shearing transformation:

$$\begin{cases} (x_{1i} - x_1) \leftarrow (x_{1i} - x_1) - m_{1i} \cdot (t_i - t) \\ (x_{2i} - x_2) \leftarrow (x_{2i} - x_1) - m_{2i} \cdot (t_i - t). \end{cases} \tag{4.8}$$

Hence, we have

$$\mathbf{H}_i^{\mathrm{m}} = \begin{bmatrix} 1 & 0 & -m_{1i} \\ 0 & 1 & -m_{2i} \\ 0 & 0 & 0 \end{bmatrix}, \tag{4.9}$$

where the detailed formulation is shown in (4.11) in the following section. Assuming a spatial prototype kernel was used with elliptical footprint, this results in a spatiotemporal kernel with the shape of a tube or cylinder with elliptical cross-sections at any time instance t. Most importantly, the center point of each such cross-section moves along the motion path.

The final component of the MASK function (4.6) is a kernel that provides temporal penalization. A natural approach is to give higher weights to samples in frames closer to the temporal position t of interest. An example of such a kernel is a Gaussian:

$$K_{h_{\mathrm{t}}}(t_i - t) = \frac{1}{h_{\mathrm{t}}} \exp\left\{ -\frac{(t_i - t)^2}{2h_{\mathrm{t}}^2} \right\}, \tag{4.10}$$

where a smoothing parameter h_{t} controls the relative temporal extent of the kernel. We use the temporal kernel (4.10) in this section to illustrate the MASK approach. However, we will introduce a more powerful adaptive temporal weighing kernel function in Section 4.2.4, which acts to compensate for unreliable local motion vector estimates.

4.2.3 Spatial Upscaling and Temporal (Frame) Interpolation

Having introduced a choice of 3-D smoothing matrix $\mathbf{H}_i$, using the Gaussian kernel for K, we have the MASK function as

$$\begin{aligned}
K^{\text{MASK}}(\mathbf{x}_i - \mathbf{x}) &= \frac{1}{\det\left(\mathbf{H}_i^{\text{s}}\right)} K\left\{\left(\mathbf{H}_i^{\text{s}}\right)^{-1} \mathbf{H}_i^{\text{m}} (\mathbf{x}_i - \mathbf{x})\right\} \cdot K_{h_{\text{t}}}(t_i - t) \\
&= \frac{1}{\det\left(\mathbf{H}_i^{\text{s}}\right)} K\left\{\left(\mathbf{H}_i^{\text{s}}\right)^{-1} \left(\mathbf{x}_i - \mathbf{x} - \begin{bmatrix} \mathbf{m}_i \\ 1 \end{bmatrix} (t_i - t)\right)\right\} \cdot K_{h_{\text{t}}}(t_i - t) \\
&= \frac{\sqrt{\det(\mathbb{C}_i)}}{h_{\text{s}}^2 h_{\text{t}}^2} \exp\left\{-\frac{1}{2h_{\text{s}}^2} \left\| \mathbf{x}_i - \mathbf{x} - \begin{bmatrix} \mathbf{m}_i \\ 1 \end{bmatrix} (t_i - t) \right\|_{\mathbb{C}_i}^2 \right\} \cdot \exp\left\{-\frac{(t_i - t)^2}{2h_{\text{t}}^2}\right\},
\end{aligned} \tag{4.11}$$

where $\|\cdot\|_{\mathbb{C}_i}^2$ is a weighted squared L_2-norm, i.e. $\|\mathbf{a}\|_{\mathbb{C}_i}^2 = \mathbf{a}^T \mathbb{C}_i \mathbf{a}$. Figure **4.1**(a-i)-(a-iii) graphically describe how the proposed MASK function (4.11) constructs its weights for spatial upscaling. For ease of explanation, suppose there are 5 consecutive frames at times from t_1 to t_5, and we upscale the third frame (spatial upscaling). When estimating the pixel value at $\mathbf{x} = [x_1, x_2, t]^T$, where $t = t_3$, first we compute $2-D$ steering kernel weights for each frame, as illustrated in Figure **4.1**(a-i), using the first Gaussian kernel function in (4.11). Motions are not taken into account at this stage. Second, having motion vectors $\mathbf{m}_i$, which we estimate using the optical flow technique with the translational motion model and the frame at $t_{i=3}$ as the anchor frame, we shift the steering kernels for each frame by $\mathbf{m}_i$ as illustrated in Figure **4.1**(a-ii). Finally, as in Figure **4.1**(a-iii), the temporal kernel function penalized the shifted kernels so that we give high weights to closer neighboring frames.

Local steering parameters and spatiotemporal weights are estimated at each pixel location $\mathbf{x}_i$ in a small cubicle for the final regression step. Once the MASK weights

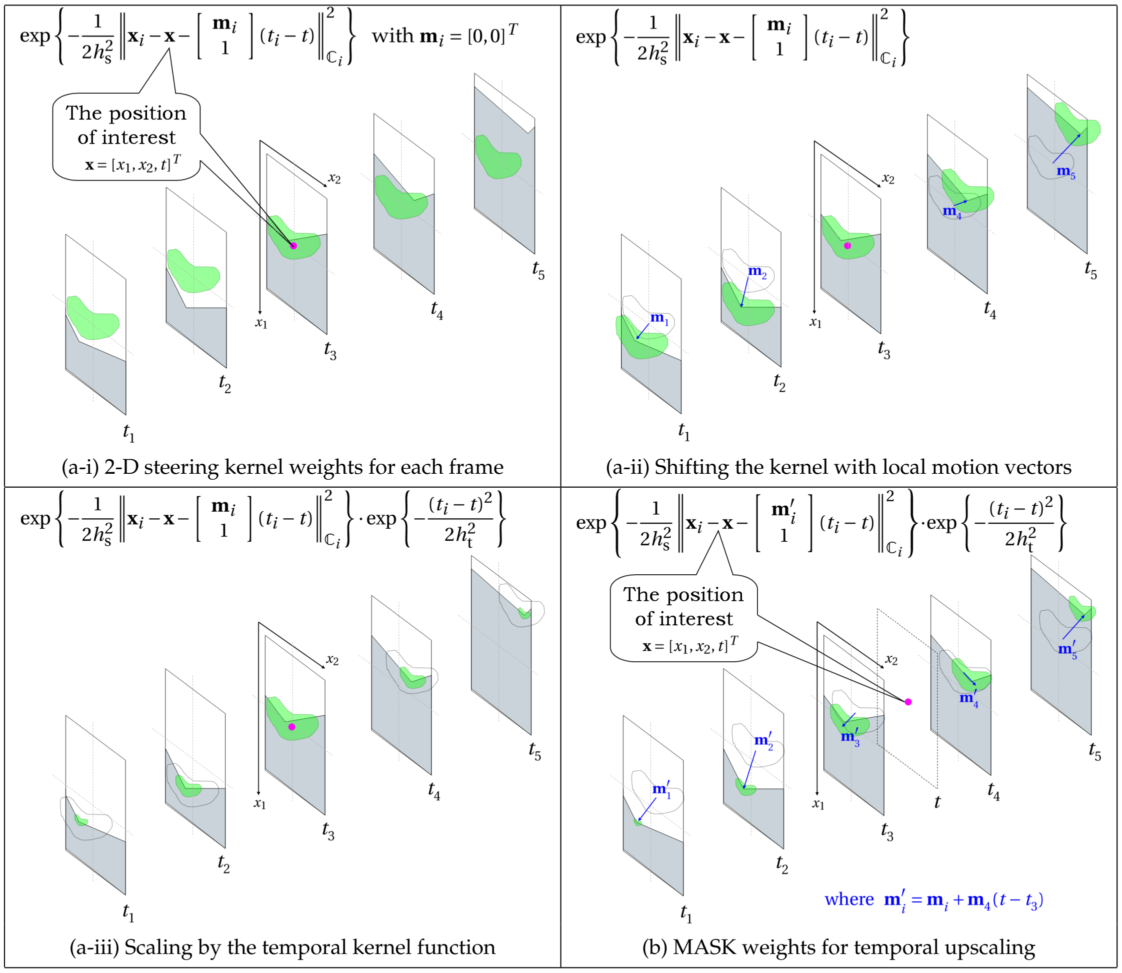

$\exp\left\{-\frac{1}{2h_s^2}\left\|\mathbf{x}_i-\mathbf{x}-\begin{bmatrix}\mathbf{m}_i\\1\end{bmatrix}(t_i-t)\right\|_{\mathbb{C}_i}^2\right\}$ with $\mathbf{m}_i=[0,0]^T$
The position of interest $\mathbf{x}=[x_1,x_2,t]^T$
x_1 x_2 t_1 t_2 t_3 t_4 t_5
(a-i) 2-D steering kernel weights for each frame
$\exp\left\{-\frac{1}{2h_s^2}\left\|\mathbf{x}_i-\mathbf{x}-\begin{bmatrix}\mathbf{m}_i\\1\end{bmatrix}(t_i-t)\right\|_{\mathbb{C}_i}^2\right\}$
$\mathbf{m}_1$ $\mathbf{m}_2$ $\mathbf{m}_4$ $\mathbf{m}_5$
(a-ii) Shifting the kernel with local motion vectors
$\exp\left\{-\frac{1}{2h_s^2}\left\|\mathbf{x}_i-\mathbf{x}-\begin{bmatrix}\mathbf{m}_i\\1\end{bmatrix}(t_i-t)\right\|_{\mathbb{C}_i}^2\right\}\cdot\exp\left\{-\frac{(t_i-t)^2}{2h_t^2}\right\}$
(a-iii) Scaling by the temporal kernel function
$\exp\left\{-\frac{1}{2h_s^2}\left\|\mathbf{x}_i-\mathbf{x}-\begin{bmatrix}\mathbf{m}'_i\\1\end{bmatrix}(t_i-t)\right\|_{\mathbb{C}_i}^2\right\}\cdot\exp\left\{-\frac{(t_i-t)^2}{2h_t^2}\right\}$
The position of interest $\mathbf{x}=[x_1,x_2,t]^T$
$\mathbf{m}'_1$ $\mathbf{m}'_2$ $\mathbf{m}'_3$ $\mathbf{m}'_4$ $\mathbf{m}'_5$ t
where $\mathbf{m}'_i=\mathbf{m}_i+\mathbf{m}_4(t-t_3)$
(b) MASK weights for temporal upscaling

Figure 4.1: Schematic representations of the construction of MASK weights. The proposed MASK weights are constructed by the following procedure: (a-i) we compute 2-D steering kernel weights for each frame (with $\mathbf{m}_i = [0,0]^T$ at this moment), (a-ii) we shift the steering kernels by the local motion vectors, and then (a-iii) we scale the shifted steering kernels by the temporal kernel function. Figure (b) illustrates the weight construction for the estimation of an intermediate frame at time t.

are available, we plug them into (4.4), compute the equivalent kernel weight function W_i, and then estimate the missing pixels and denoise the given samples from the local input samples y_i around the position of interest $\mathbf{x}$. Similar to (4.5), the final spatiotemporal regression step can be expressed as follows:

$$\hat{z}^{\mathrm{MASK}}(\mathbf{x}) = \sum_{i=1}^{P} W_i(\mathbf{x}, \mathbf{H}_i^{\mathrm{s}}, \mathbf{H}_i^{\mathrm{m}}, h_{\mathrm{t}}, K, N)\, y_i. \tag{4.12}$$

The MASK approach is also capable of upscaling video temporally (also called frame interpolation or frame rate upconversion). Figure **4.1**(b) illustrates the MASK weights for estimating an intermediate frame at sometime between the third frame at t_3 and the fourth frame at t_4. Fundamentally, following the same procedure as described in Figures **4.1**(a-i)-(a-iii), we generate MASK weights. However, for the motion vector with the unknown intermediate frame as the anchor frame, we assume that the motion between the frames at t_3 and t_4 is constant, and using the motion vectors $\mathbf{m}_i$ for $i = 1, \cdots, 5$, we linearly interpolate motion vectors $\mathbf{m}'_i$ as

$$\mathbf{m}'_i = \mathbf{m}_i + \mathbf{m}_4(t - t_3), \quad \text{where } t_3 \le t < t_4. \tag{4.13}$$

Note that when $\mathbf{m}_4$ is inaccurate, the interpolated motion vectors for other frames in the temporal window ($\mathbf{m}'_i$) are also inaccurate. In this case, we would shift the kernel toward the wrong direction, and the MASK weights would be less effective for temporal upscaling. Therefore, one should incorporate a test of the reliability of $\mathbf{m}_4$ into the process, and use vectors $\mathbf{m}_i$ instead of $\mathbf{m}'_i$ if it is found to be unreliable. Our specific technique to compute the reliability of motion vectors in described next in Section 4.2.4.

4.2.4 A Practical Video Upscaling Algorithm Based on MASK

In this section, we describe a complete algorithm for spatial upscaling, denoising and enhancement, as well as temporal frame interpolation, based on the MASK approach. We introduce several techniques that enable a practical implementation of the MASK principles explained in the previous section. In particular, we develop an algorithm with reduced computational complexity and reduced memory requirements, that is suitable for both software and hardware implementation.

An overview of the proposed video interpolation and denoising algorithm based on the MASK approach is provided in Figure **4.2**. The algorithm estimates spatial steering parameters and motions using (2.8) with (2.13) and the optical flow technique (q.v. Appendix D). Hence, we first compute initial estimates of the spatial and temporal derivatives, e.g. based on classic kernel regression. In this work, we obtain quick and robust estimate of the spatial orientation angle, elongation and scaling parameters, θ_i, ϱ_i, and γ_i, at $\mathbf{x}_i$ by applying a vector quantization technique [83] to the covariance matrix obtained from the spatial gradient data as shown in (2.13). Motion vectors are estimated using the well-known Lucas and Kanade's method, first introduced in [84] (see Appendix D for our detailed implementation), based on both spatial and temporal gradients in a local region. This is followed by computing estimates of the temporal motion reliability (η), and will be described further in this section. Given spatial and motion steering parameters, final MASK regression is applied directly on the input video samples.

The following are further salient points for our algorithm based on MASK. We first summarize them, and then provide details in subsequent subsections.

- ▷ **Block-by-Block Processing**

  Since the kernel-based estimator is a pointwise process, it is unnecessary to store the orientations and motion vectors of all the pixels in a video frame ($\mathbf{H}_i^{\mathrm{s}}$ and $\mathbf{H}_i^{\mathrm{m}}$ for all i) in memory. However, strict pixel-by-pixel processing would result in a large number of redundant computations due to the overlapping neighborhoods

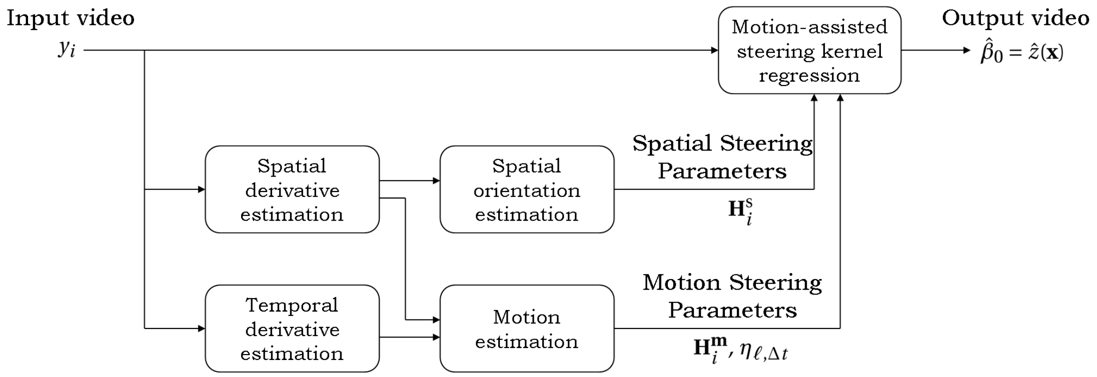

Input video
y_i
Motion-assisted steering kernel regression
Output video
$\hat{\beta}_0 = \hat{z}(\mathbf{x})$
Spatial derivative estimation
Spatial orientation estimation
Spatial Steering Parameters
$\mathbf{H}_i^{\mathrm{s}}$
Temporal derivative estimation
Motion estimation
Motion Steering Parameters
$\mathbf{H}_i^{\mathbf{m}}, \eta_{\ell,\Delta t}$

Figure 4.2: Illustration of spatiotemporal video upscaling based on the MASK approach.

of nearby pixels. In order to reduce the computational load while keeping the required memory space small, we break the video data into small blocks (e.g. 8×8 pixels), and process the blocks one-by-one.

▷ **Adaptive Temporal Penalization**

MASK relies on motion vectors, and the visual quality of output video frames is strongly associated with the accuracy of motion estimation. Even though our motion estimation approach is able to estimate motion vectors quite accurately, the estimated vectors become unreliable when the underlying scene motion and camera projection violate the motion model. In practice, errors in motion vectors are inevitable and it is important to provide a fall-back mechanism in order to avoid visual artifacts.

▷ **Quantization of Orientation Map**

The estimation of spatial orientations or steering covariance matrices $\mathbf{C}_i^{\mathrm{s}}$ by (2.13) involves singular value decomposition (SVD), which represents significant computational complexity. Instead of using the SVD, we use a pre-defined lookup table containing a set of candidate covariance matrices, and locally select an appropriate matrix from the table. Since the lookup table contains only stable (invertible) covariance matrices, the estimation process remains robust.

▹ **Adaptive Regression Order**

A higher regression order (e.g. $N = 2$ in this chapter) preserves high frequency components in filtered images, although it requires more computation due to the number of bases (column vectors) in $\mathbf{X}$ (1.37). On the other hand, zeroth regression order ($N = 0$) has lower computational cost, but it has a stronger smoothing effect. Although second order regression is preferable, it is only needed at pixel locations in texture and edge regions. Moreover, in terms of noise reduction, zeroth order regression is more suitable in flat regions. We propose an approximation to adjust the order N locally, based on the scaling parameter (γ_i). Consequently, this adaptive approach keeps the total computational cost low while it preserves, and even enhances, high frequency components.

4.2.4.1 Block-by-Block Processing

The overall MASK algorithm consists of several operations (i.e. estimating spatial and temporal gradients, spatial orientations, and motions as shown in Figure **4.2** and finally applying kernel regression), and it is possible to implementation these in, for example, a pixel-by-pixel process or a batch process. For the pointwise process, we estimate gradients, orientations, and motions individually, and then finally estimate a pixel value. Note that most of these operations require calculations involving other pixels in a neighborhood around the pixel of interest. Since the neighborhoods may overlap significantly, frequently the same calculation would be performed multiple times. Hence, a pointwise implementation suffers from a large computational load. On the other hand, this implementation requires very little memory. On the other hand, for the batch process, we estimate gradients for all pixels in an entire frame and store the results in memory, then estimate orientations of all the pixels and store those results, etc. In the batch implementation, we need a large amount of memory space to store intermediate results for all the pixels in a frame: however, it avoids repeated calculations. This type of process is impractical particularly for a hardware implementation.

As a compromise, in order to limit both the computational load and the use of memory, we process a video frame in a block-by-block manner, where each block contains, for instance, 8×8 or 16×16 pixels. Further reduction of the computational load is achieved by using a block-based motion model: we assume that, within a block, the motion of all the pixels follow a parametric model, e.g., translational or affine. In this chapter, we fix the block size to 8×8 pixels and we use the translational motion model.

4.2.4.2 Motion Estimation and Adaptive Temporal Penalization

Motion estimation is based on the well-known Lucas and Kanade's method [84, 85], applied in a block-by-block manner as follows. Assume we computed initial estimates of the local spatial and temporal derivatives. For example, spatial derivatives may be computed using classic kernel regression or existing derivative filtering techniques, e.g. Sobel filter [29]. Temporal derivatives are computed by taking the temporal difference between pixels of the current frame and one of the neighboring frames. Let $\hat{\underline{\mathbf{z}}}_{x_1,\ell}$, $\hat{\underline{\mathbf{z}}}_{x_2,\ell}$, and $\hat{\underline{\mathbf{z}}}_{t,\ell}$ denote vectors containing (in lexicographical order) derivative estimates from the pixels in a local analysis window ω associated with the ℓ-th block in the frame. This window contains and is typically centered on the block of pixels of interest, but may include additional pixels beyond the block (i.e. analysis windows from neighboring blocks may overlap). A motion vector $\mathbf{m}_\ell$ for the ℓ-th block is estimated by solving the optical flow equation ($[\underline{\hat{\mathbf{z}}}_{x_1,\ell}, \underline{\hat{\mathbf{z}}}_{x_2,\ell}]\mathbf{m}_\ell + \underline{\hat{\mathbf{z}}}_{t,\ell} = [0, \cdots, 0]^T$) in the least-squares sense. The basic Lucas and Kanade's method is applied iteratively for improved performance. As explained before, MASK uses multiple frames in a temporal window around the current frame. For every block in the current frame, a motion vector is computed to each of the neighboring frames in the temporal window. Hence, if the temporal window contains 4 neighboring frames in addition to the current frame, we compute 4 motion vectors for each block in the current frame.

In practice, a wide variety of transitions/activies will occur in natural video. Some of these are so complex that no parametric motion model matches them exactly, and motion errors are unavoidable. When there are errors in the estimated motion vectors, visually unacceptable artifacts may be introduced in the reconstructed frames due to the motion-based processing. One way to avoid such visible artifacts in upscaled frames is to adapt the temporal weighting based on the correlation between the current block and the corresponding blocks in other frames determined by the motion vectors. That is to say, before constructing MASK weights, we compute the reliability (η_ℓ) of each estimated motion vector. A simple way to define η_ℓ is to use the mean square error or mean absolute error between the block of interest and the corresponding block in the neighboring frame towards which the motion vector is pointing. Once the reliability of the estimated motion vector is available, we penalize the steering kernels by a temporal kernel K_t, a kernel function of η. Figure **4.3** illustrates the temporal

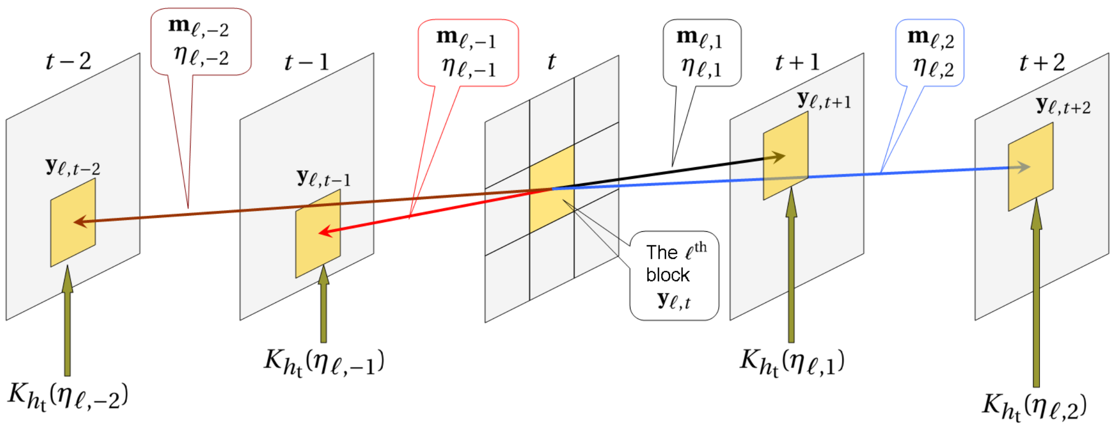

$t-2$
$\mathbf{m}_{\ell,-2}$ $\eta_{\ell,-2}$
$t-1$
$\mathbf{m}_{\ell,-1}$ $\eta_{\ell,-1}$
t
$\mathbf{m}_{\ell,1}$ $\eta_{\ell,1}$
$t+1$
$\mathbf{m}_{\ell,2}$ $\eta_{\ell,2}$
$t+2$
$\mathbf{y}_{\ell,t-2}$
$\mathbf{y}_{\ell,t-1}$
$\mathbf{y}_{\ell,t+1}$
$\mathbf{y}_{\ell,t+2}$
The ℓ^{th} block $\mathbf{y}_{\ell,t}$
$K_{h_t}(\eta_{\ell,-2})$
$K_{h_t}(\eta_{\ell,-1})$
$K_{h_t}(\eta_{\ell,1})$
$K_{h_t}(\eta_{\ell,2})$

Figure 4.3: A schematic representation of temporal weighting in MASK for upscaling the ℓ-th block ($\mathbf{y}_{\ell,t}$) of the frame at time t. First, we locate the neighboring blocks ($\mathbf{y}_{\ell,i}$ for $i = -2, -1, 1, 2$) indicated by the motion vectors ($\mathbf{m}_{\ell,i}$). Then, we compute the motion reliability ($\eta_{\ell,i}$) based on the difference between the ℓ-th block at t and the neighboring blocks, and combine the temporal penalization by K_t with the spatial kernel function K.

weighting, incorporating motion reliability. Suppose we upscale the ℓ-th block in the frame at time t using 2 previous and 2 forward frames, and there are 4 motion vectors, $\mathbf{m}_{\ell,i}$, between a block in the frame at t and the 4 neighboring frames. First, we find the blocks that the motion vectors indicate from the neighboring frames shown as $\mathbf{y}_{\ell,i}$ in Figure **4.3**. Then, we compute the motion reliability based on the difference between the ℓ-th block at t and other blocks and decide the temporal penalization for each neighboring block.

More specifically, we define the motion reliability $\eta_{\ell,\Delta t}$ and the adaptive temporal kernel as

$$\eta_{\ell,\Delta t} = \frac{\left\| \underline{\mathbf{y}}_{\ell,t} - \underline{\mathbf{y}}_{\ell,t+\Delta t} \right\|_2}{M}, \tag{4.14}$$

$$K_{h_t}(\eta_{\ell,\Delta t}) = \frac{1}{1 + \frac{\eta_{\ell,\Delta t}}{h_t}} \tag{4.15}$$

where h_t is the (global) temporal smoothing parameter, that controls the strength of the temporal penalization, $\underline{\mathbf{y}}_{\ell,t}$ is the ℓ-th block of the frame at time t (in lexicographical order), $t+\Delta t$ is the temporal position of a neighboring frame, and M is the total number of pixels in a block. We replace the temporal kernel in (4.11) by (4.15). This temporal weighting technique is similar to the adaptive weighted averaging (AWA) approach proposed in [86]; however, the weights in AWA are computed pointwise. In MASK, the temporal kernel weights are a function of radiometric distances between small blocks and are computed block-wise.

4.2.4.3 Quantization of Orientation Map

The computational cost of estimating local spatial steering (covariance) matrices is high due to the SVD. In this section, using the well-known technique of *vector quantization* [83], we describe a way to obtain stable (invertible) steering matrices without using the SVD. Briefly speaking, first, we construct a look-up table which has a certain number of stable (invertible) steering matrices. Second, instead of computing the steering matrix by (2.13), we compute the naive covariance matrix (2.10), and then find the most similar steering matrix from the look-up table. The advantages of using the look-up table are that (i) we can lower the computational complexity by avoiding singular value decomposition, (ii) we can control and trade-off accuracy and computational load by designing an appropriate vector quantization scheme with almost any desired number of steering matrices in the look-up table, and (iii) we can pre-calculate kernel weights to lower the computational load further (since the steering matrices are fixed).

From (2.13), the elements of the spatial covariance matrix $\mathbf{C}_i^s$ are given by the steering parameters with the following equations:

$$\mathbf{C}_i^{\mathrm{s}}(\gamma_i, \varrho_i, \theta_i) = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}, \tag{4.16}$$

with

$$c_{11} = \gamma_i \left(\varrho_i \cos^2\theta_i + \varrho_i^{-1} \sin^2\theta_i\right), \tag{4.17}$$

$$c_{12} = c_{21} = -\gamma_i \left(\varrho_i + \varrho_i^{-1}\right) \cos\theta_i \sin\theta_i, \tag{4.18}$$

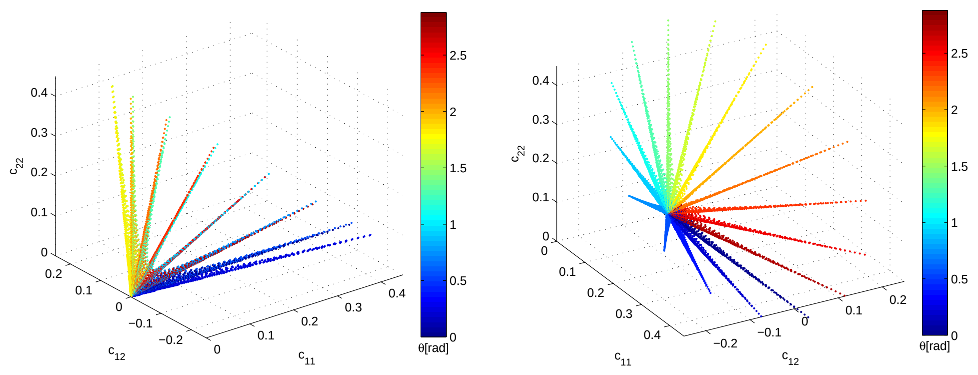
$$c_{22} = \gamma_i \left(\varrho_i \sin^2\theta_i + \varrho_i^{-1} \cos^2\theta_i\right), \tag{4.19}$$

where, again, γ_i is the scaling parameter, ϱ_i is the elongation parameter, and θ_i is the orientation angle parameter for the pixel y_i. Figure **4.4** visualizes the relationship between the steering parameters and the elements of the covariance matrix. Based on the above formulae, using a pre-defined set of the scaling, elongation, and angle parameters, we can can generate a lookup table for covariance matrices, off-line.

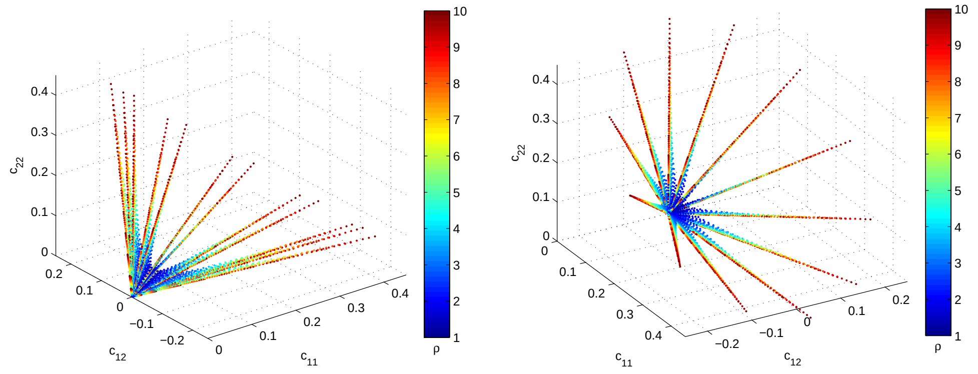
During the on-line processing stage, we compute a naive covariance matrix $\mathbf{C}_i^{\mathrm{naive}}$ (2.10) and then normalize $\mathbf{C}_i^{\mathrm{naive}}$ so that the determinant of the normalized covariance matrix $\widetilde{\mathbf{C}}_i^{\mathrm{naive}}$ equals 1:

$$\widetilde{\mathbf{C}}_i^{\mathrm{naive}} = \frac{\mathbf{C}_i^{\mathrm{naive}}}{\sqrt{\det \mathbf{C}_i^{\mathrm{naive}}}} = \frac{1}{\gamma_i} \mathbf{C}_i^{\mathrm{naive}}. \tag{4.20}$$

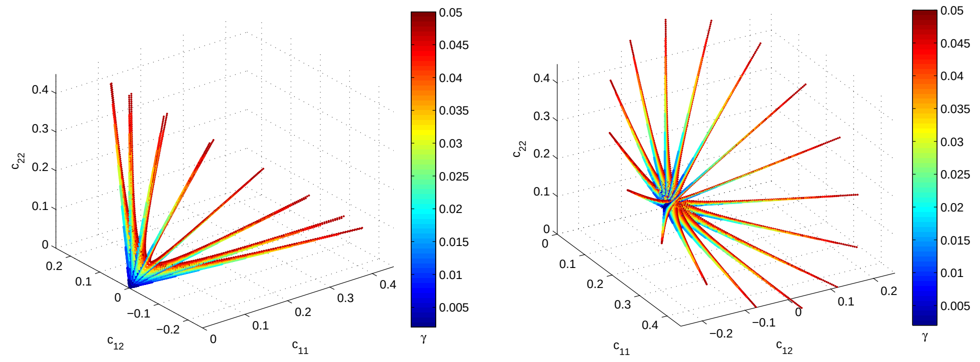
This normalization eliminates the scaling parameter from the lookup table and simplifies the relationship between the elements of the covariance matrix and the rest of the steering parameters (i.e. ϱ_i and θ_i), and allows us to reduce the size of the table. Table **4.1** shows an example of a compact lookup table. When the elongation parameter ϱ_i is smaller than 2.5, we decide that there is no significant edge structure at the position $\mathbf{x}_i$ and $\widetilde{\mathbf{C}}_i^{\mathrm{naive}}$ is quantized as an identity matrix (i.e. the kernel spreads equally



(a) varying the angle parameter



(b) varying the elongation parameter



(c) varying the scaling parameter

Figure 4.4: The graphical relationship between the steering kernel parameters and the values of covariance matrix.

every direction). Note that, after the quantization, since we put the effect of the scaling parameter γ_i back into the covariance matrix, the width of the kernel contour stays

Table 4.1: A compact lookup table for the normalized covariance matrix $\widetilde{\mathbf{C}}_j^{\mathrm{s}}(\varrho_j, \theta_j) = [\tilde{c}_{11}, \tilde{c}_{12}; \tilde{c}_{12}, \tilde{c}_{22}]$.

$\tilde{c}_{11}$	$\tilde{c}_{12}$	$\tilde{c}_{22}$	ϱ_j	θ_j
1.0000	0	1.0000	1.0	0
2.5000	0	0.4000	2.5	0
2.1925	1.0253	0.7075	2.5	$\frac{1}{8}\pi$
1.4500	1.4500	1.4500	2.5	$\frac{2}{8}\pi$
0.7075	1.0253	2.1925	2.5	$\frac{3}{8}\pi$
0.4000	0	2.5000	2.5	$\frac{4}{8}\pi$
0.7075	-1.0253	2.1925	2.5	$\frac{5}{8}\pi$
1.4500	-1.4500	2.1925	2.5	$\frac{6}{8}\pi$
2.1925	-1.0253	0.7075	2.5	$\frac{7}{8}\pi$

small at texture regions. On the other hand, when $\varrho_i \geq 2.5$, we quantize $\widetilde{\mathbf{C}}_i^{\mathrm{naive}}$ with 8 angles. Using $\widetilde{\mathbf{C}}_i^{\mathrm{naive}}$, we obtain the closest covariance matrix $\widetilde{\mathbf{C}}_i^{\mathrm{s}}$ from the table by

$$\widetilde{\mathbf{C}}_i^{\mathrm{s}} = \underset{\widetilde{\mathbf{C}}_j^{\mathrm{s}}(\varrho_j, \theta_j)}{\arg\min} \left\| \widetilde{\mathbf{C}}_j^{\mathrm{s}}(\varrho_j, \theta_j) - \widetilde{\mathbf{C}}_i^{\mathrm{naive}} \right\|_{\mathrm{F}}, \tag{4.21}$$

where $\|\cdot\|_{\mathrm{F}}$ is the Frobenius norm. Finally, putting the scaling parameter back into the covariance matrix, we have the quantized covariance matrix as

$$\widehat{\mathbf{C}}_i^{\mathrm{s}} = \gamma_i \widetilde{\mathbf{C}}_i^{\mathrm{s}}. \tag{4.22}$$

4.2.4.4 Adaptive Regression Order

The advantage of a higher regression order is that the estimated images preserve high frequency components, though the higher order regressor requires more computational load. In this section, we discuss how we can reduce the computational complexity, while enabling adaptation of the regression order. According to [87], the second order equivalent kernel $\mathbf{w}_0(2)$ in classic KR (1.37) can be obtained approximately from the zeroth order one $\mathbf{w}_0(0)$ in (1.37) as follows:

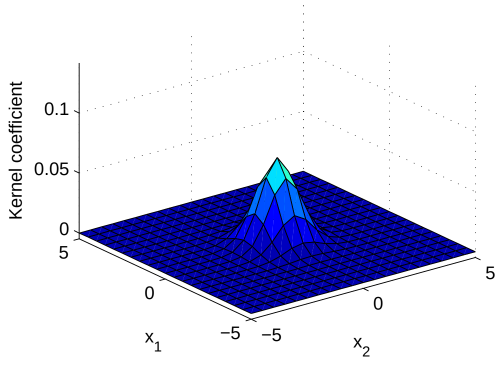
$$\mathbf{w}_0^T(2) \approx \widetilde{\mathbf{w}}_0^T(2) = \mathbf{w}_0^T(0) - \kappa \boldsymbol{\Gamma} \mathbf{w}_0^T(0), \tag{4.23}$$

where $\boldsymbol{\Gamma}$ is Laplacian kernel[2] in matrix form (we use $[1, 1, 1; 1, -8, 1; 1, 1, 1]$ as a discrete Laplacian kernel) and κ is a regression order adaptation parameter. This operation can be seen to "sharpen" the equivalent kernel, and is equivalent to sharpening the reconstructed image. Figure **4.5** shows the comparison between the actual second order equivalent kernel $\mathbf{w}_0(2)$ and the approximated equivalent kernel $\widetilde{\mathbf{w}}_0(2)$ given by (4.23). In the comparison, we used the Gaussian function for K, and compute the zeroth order and the second order equivalent kernels, shown in Figures **4.5**(a) and (b), respectively. The approximated equivalent kernel $\widetilde{\mathbf{w}}_0(2)$ is shown in Figure **4.5**(c), and Figure **4.5**(d) compares the horizontal cross section of $\mathbf{w}_0(0)$, $\mathbf{w}_0(2)$, and $\widetilde{\mathbf{w}}_0(2)$. As seen in Figure **4.5**(d), $\widetilde{\mathbf{w}}_0(2)$ is close to the exact one $\mathbf{w}_0(2)$.

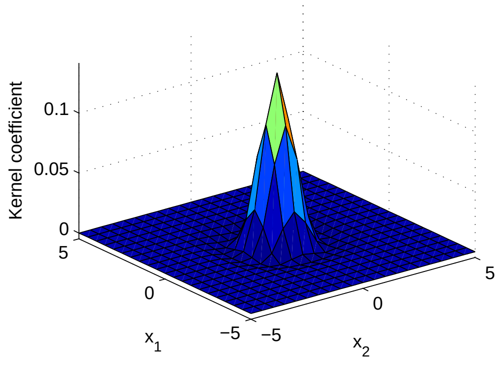
There are two advantages brought by the approximation (4.23): (i) The formula simplifies the computation of the second order equivalent kernels, i.e., we no longer need to generate the basis matrix $\mathbf{X}$ or take inversion of matrices. (ii) Since the effect of the second order regression is now explicitly expressed by $\kappa \boldsymbol{\Gamma} \mathbf{w}_0^T(0)$ in (4.23), the formulation allows for adjustment of the regression order across the image, but also it allows for "fractional" regression orders, providing fine control over the amount of sharpening (κ) applied locally.

We propose a technique to automatically select the regression order parameter κ adaptively as follows. By setting κ near zero in flat regions and to a large value in edge and texture regions, we can expect a reduction of computational complexity, pre-

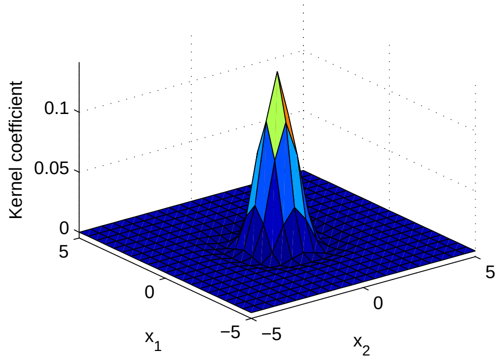
[2] Laplacian of Gaussian (LoG) kernel makes the approximation (4.23) more accurate.



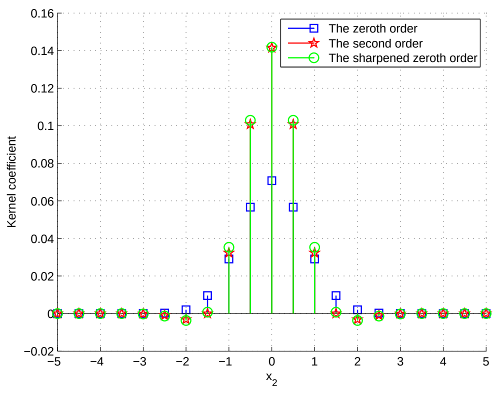
(a) The zeroth order equivalent kernel ($\mathbf{w}_0(0)$)



(b) The second order equivalent kernel ($\mathbf{w}_0(2)$)



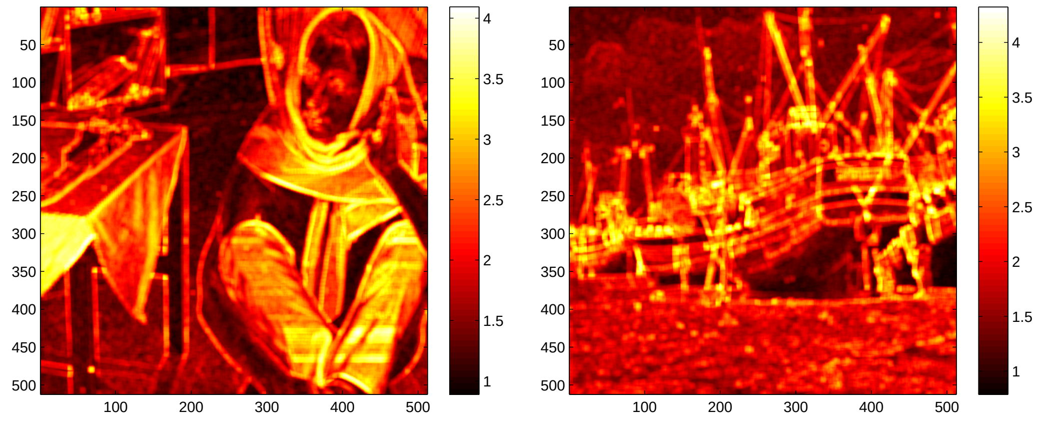
(c) A sharpened zeroth order equivalent kernel ($\widetilde{\mathbf{w}}_0(2)$)



(d) Horizontal cross sections of the equivalent kernels

Figure 4.5: Equivalent kernels given by classic kernel regression: (a) the zeroth order equivalent kernel with the global smoothing parameter $h = 0.75$, (b) the second order equivalent kernel ($\mathbf{w}_0(2)$) with $h = 0.75$, (c) a sharpened zeroth order equivalent kernel ($\widetilde{\mathbf{w}}_0(2)$) with a 3×3 Laplacian kernel ($\mathbf{\Gamma} = [1,\ 1,\ 1;\ 1,\ -8,\ 1;\ 1,\ 1,\ 1]$) and $\kappa = 0.045$, and (d) Horizontal cross sections of the equivalent kernels $\mathbf{w}_0(0)$, $\mathbf{w}_0(2)$, and $\widetilde{\mathbf{w}}_0(2)$. For this example, we used a Gaussian function for $K(\cdot)$.

vent amplifying noise component in flat regions, and preserve or even enhance texture regions and edges. In order to select spatially adapted regression factors, we can make use of the scaling parameter γ_i, which we earlier used to normalize the covariance matrix in (4.20). This makes practical sense since γ_i is high in texture and edge areas and low in flat areas as shown in Figure **4.6**. Because γ_i is already computed when comput-



(a) Barbara (b) Boat

Figure 4.6: Local scaling parameters (γ_i) for (a) Barbara image and (b) Boat image. With the choice of the adaptive regression order $\kappa_i = 0.01\gamma_i$ (4.24), the regression order becomes nearly zero in the areas where γ_i is close to zero, while in areas where γ_i is around 5, the resulting equivalent kernel given by (4.23) approximately becomes second order.

ing the steering matrices, no extra computation is required. A good way to choose the regression factor κ locally is to make it a simple function of γ_i. Specifically, we choose our adaptive regression factor by

$$\kappa_i = 0.01\gamma_i, \tag{4.24}$$

where 0.01 is a global parameter controlling the overall sharpening amount. For instance, it is possible to choose a larger number if a stronger sharpening effect is desired globally. As shown in Figure **4.6**, with the choice of the adaptive regression order as in (4.24), the regression order becomes close to zero in the area where γ_i is close to zero, while the resulting equivalent kernel given by (4.23) approximately becomes a second order kernel in the area where γ_i is around 5. Setting κ too large results in overshoot of pixel values around texture and edges. We process color video in the YCbCr domain and estimate spatial orientations in the luminance component only, since the human visual system is most sensitive to orientations in the luminance component.

4.3 3-D Steering Kernel Regression

A direct extension of the SKR framework introduced in Chapter 2 to 3-D signals for the express purpose of video denoising and resolution enhancement is the next subject of this chapter. As we shall see, since the development in 3-D involves the computation of orientation in space-time [87], motion information is implicitly and reliably captured. Therefore, unlike conventional approaches to video processing including the MASK approach presented in the previous section, 3-D SKR does not require explicit estimation of (modestly sized but essentially arbitrarily complex) motions, as this information is implicitly captured with in the locally "learned" metric. It is worth mentioning in passing here that the approach we take, while independently derived, is in the same spirit as the body of work known as *metric learning* in the machine learning community, e.g. [88].

Naturally, the performance of the proposed approach is closely correlated with the quality of estimated spatiotemporal orientations. In the presence of noise, aliasing, and other artifacts, the estimates of orientations may not be initially accurate enough, and as we explain in Section 4.3.1, we therefore propose an iterative mechanism for estimating the orientations, which relies on the estimate of the pixels from the previous iteration similar to the 2-D case described in Figure **2.12**.

To be more specific, as shown in Figure **4.11**, we can first process a video sequence with orientation estimates of modest quality. Next, using the output of this first step, we can re-estimate the spatiotemporal orientations, and repeat this process several times. As this process continues, the orientation estimates are improved, as is the quality of the output video. The overall algorithm we just described will be referred to as the 3-D iterative steering kernel regression (3-D ISKR).

As we will see in the coming sections, the approach we introduce here is ideally suited for implicitly capturing relatively small motions using the orientation tensors. However, if the motions are somewhat large, the resulting (3-D) local similarity measure, due to its inherent local nature, will fail to find similar pixels in nearby frames. As a result, the 3-D kernels essentially collapse to become 2-D kernels centered around the pixel of interest within the same frame. Correspondingly, the net effect of the algorithm would be to do frame-by-frame 2-D upscaling. For such cases, as discussed in Section 4.3.2, some level of explicit motion estimation is unavoidable in order to reduce temporal aliasing and achieve resolution enhancement. This rough motion estimate can then be used to "neutralize" or "compensate" for the large motion, leaving behind a residual of small motions, which can be implicitly captured within the 3-D orientation kernel. In summary, the 3-D SKR approach can accommodate a variety of complex motions in the input videos by a two-tiered approach: (i) large displacements are neutralized by rough motion compensation either globally or block-by-block as appropriate, and (ii) 3-D ISKR handles the fine-scale and detailed rest of the possibly complex motion present.

4.3.1 Spatiotemporal Steering Kernel regression

The 3-D extension of the SKR framework is straightforward. For the minimization of 3-D kernel regression (4.4), similar to the 2-D SKR case, we define the steering matrix $\mathbf{H}_i$ as

$$\mathbf{H}_i \equiv h\,(\mathbf{C}_i)^{-\frac{1}{2}}, \tag{4.25}$$

where the covariance matrix $\mathbf{C}_i \in \mathscr{R}^{3\times 3}$ can be naively estimated as

$$\mathbf{C}_i = \mathbf{J}_i^T \mathbf{J}_i \tag{4.26}$$

with

$$\mathbf{J}_i = \begin{bmatrix} \vdots & \vdots & \vdots \\ z_{x_1}(\mathbf{x}_j) & z_{x_2}(\mathbf{x}_j) & z_t(\mathbf{x}_j) \\ \vdots & \vdots & \vdots \end{bmatrix}, \quad \mathbf{x}_j \in \xi_i, \quad j = 1, \cdots, Q, \tag{4.27}$$

where $z_{x_1}(\cdot)$, $z_{x_2}(\cdot)$, and $z_t(\cdot)$ are the first derivatives along x_1-, x_2- and t-axes, ξ_i is a local analysis cubicle around a sample position at $\mathbf{x}_i$, and Q is the number of rows in $\mathbf{J}_i$. Once again for the sake of robustness, as explained in Section 2.1.2, we compute a more stable estimate of $\mathbf{C}_i$ by invoking the SVD of $\mathbf{J}_i$ with regularization as:

$$\widehat{\mathbf{C}}_i = \gamma_i \sum_{q=1}^{3} \varrho_q \mathbf{v}_q \mathbf{v}_q^T, \tag{4.28}$$

with

$$\begin{aligned} \varrho_1 &= \frac{s_1 + \lambda'}{s_2 s_3 + \lambda'}, \quad \varrho_2 = \frac{s_2 + \lambda'}{s_1 s_3 + \lambda'}, \\ \varrho_3 &= \frac{s_3 + \lambda'}{s_1 s_2 + \lambda'}, \quad \gamma_i = \left(\frac{s_1 s_2 s_3 + \lambda''}{Q} \right)^{\alpha}, \end{aligned} \tag{4.29}$$

where ϱ_q and γ_i are the *elongation* and *scaling* parameters, respectively, λ' and λ'' are regularization parameters that dampen the noise effect and restrict γ_i, the denominators of ϱ_q's from being zero (q.v. Appendix E for the derivations), and Q is the number of rows in $\mathbf{J}_i$. We fix $\lambda' = 1$ and $\lambda'' = 0.1$ throughout this work. The singular values (s_1, s_2, and s_3) and the singular vectors ($\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$) are given by the (compact) SVD of $\mathbf{J}_i$:

$$\mathbf{J}_i = \mathbf{U}_i \mathbf{S}_i \mathbf{V}_i^T = \mathbf{U}_i \operatorname{diag}\{s_1, s_2, s_3\} \left[\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\right]^T. \tag{4.30}$$

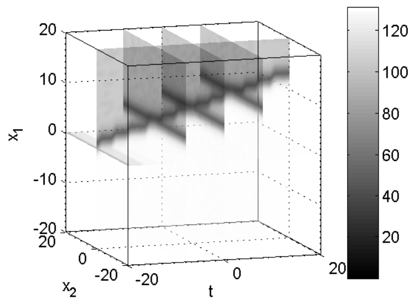
similar to the 2-D case, the steering kernel function in 3-D is defined as

$$K_{\mathbf{H}_i}(\mathbf{x}_i - \mathbf{x}) = \sqrt{\frac{\det(\mathbf{C}_i)}{(2\pi h^2)^3}} \exp\left\{ -\frac{1}{2h^2} \left\| (\mathbf{C}_i)^{\frac{1}{2}} (\mathbf{x}_i - \mathbf{x}) \right\|_2^2 \right\}, \tag{4.31}$$

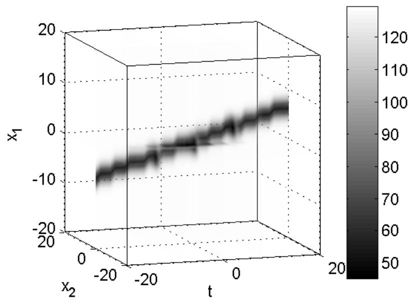
with $\mathbf{x} = [x_1, x_2, t]$. The main tuning parameters are the global smoothing parameter (h) in (4.31) and the structure sensitivity (α) in (4.29). The specific choices of these parameters are indicated in Section 4.4, and Section 2.1.2 gives more details about the parameters h and α.

Figure **4.7** shows visualizations of 3-D weights given by the steering kernel function for two cases: (a) a horizontal edge moving vertically over time (creating a tilted plane in the local cubicle), and (b) a small circular dot also moving vertically over time (creating a thing tube in a local cubicle). Considering the case of denoising for the pixel located at the center of each data cube of Figures **4.7**(a) and (b), we have the steering kernel weights illustrated in Figures **4.7**(c)(d) and (e)(f). Figures **4.7**(c)(d) and (e)(f) show the cross-sections and the isosurface of the weights, respectively. As seen in these figures, the weights faithfully reflect the local signal structure in space-time. Also, Figure **4.8** gives a graphical representation of the 3-D steering kernel weights for the Foreman sequence. In the figure, we show the cross sections (transverse, sagittal, and axial) of the video (3-D) data, and draw the cross sections of the steering kernel weights at every 15 pixels in every direction. For this example, we chose the analysis cubicle sizes $\omega = 15 \times 15 \times 15$ and $\xi_i = 5 \times 5 \times 5$. It is worth noting that the orientation structures which appear in the x_1-t and x_2-t cross sections are motion trajectories, and our steering kernel weights fit the local motion trajectories without explicit motion estimation.

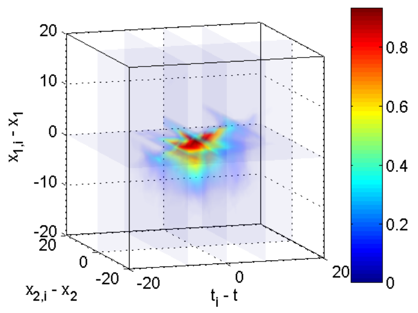
As illustrated in Figures **4.7** and **4.8**, the weights provided by the steering kernel function capture the local signal structures which include both spatial and temporal edges. Here we give a brief description of how orientation information thus captured in 3-D contains the motion information implicitly. It is convenient in this respect to use the (gradient-based) optical flow framework [50, 89, 90] to describe the underlying idea. Defining the 3-D motion vector as $\widetilde{\mathbf{m}}_i = [m_1, m_2, 1]^T = [\mathbf{m}_i^T, 1]^T$ and invoking the



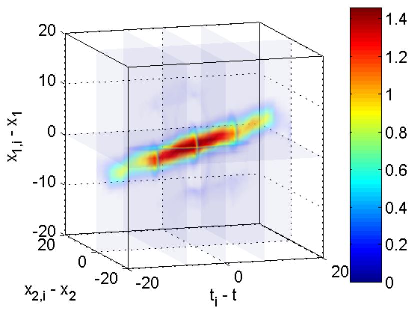
(a) A tilted plane



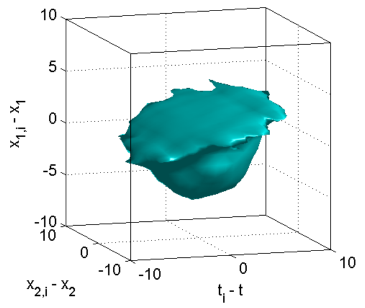
(b) A thin tube



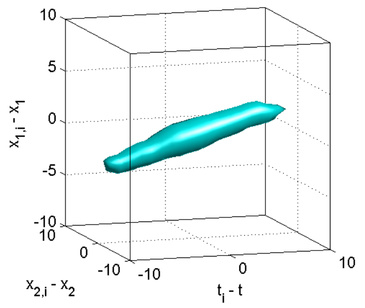
(c) Cross-sections of SK weights of (a)



(d) Cross-sections of SK weights of (b)



(e) The isosurface of (c)



(f) The isosurface of (d)

Figure 4.7: Visualizations of steering kernels for (a) the case of one horizontal edge moving up (this creates a tilted plane in a local cubicle) and (b) the case of one small dot moving up (this creates a thin tube in a local cubicle). (a) and (b) show some cross-sections of the 3-D data, and (b) and (c) show the cross-sections of the weights given by the steering kernel function when we denoise the sample located at the center of the data cube, and (d) and (e) show the isosurface of the steering kernel weight for (a) and (b), respectively.

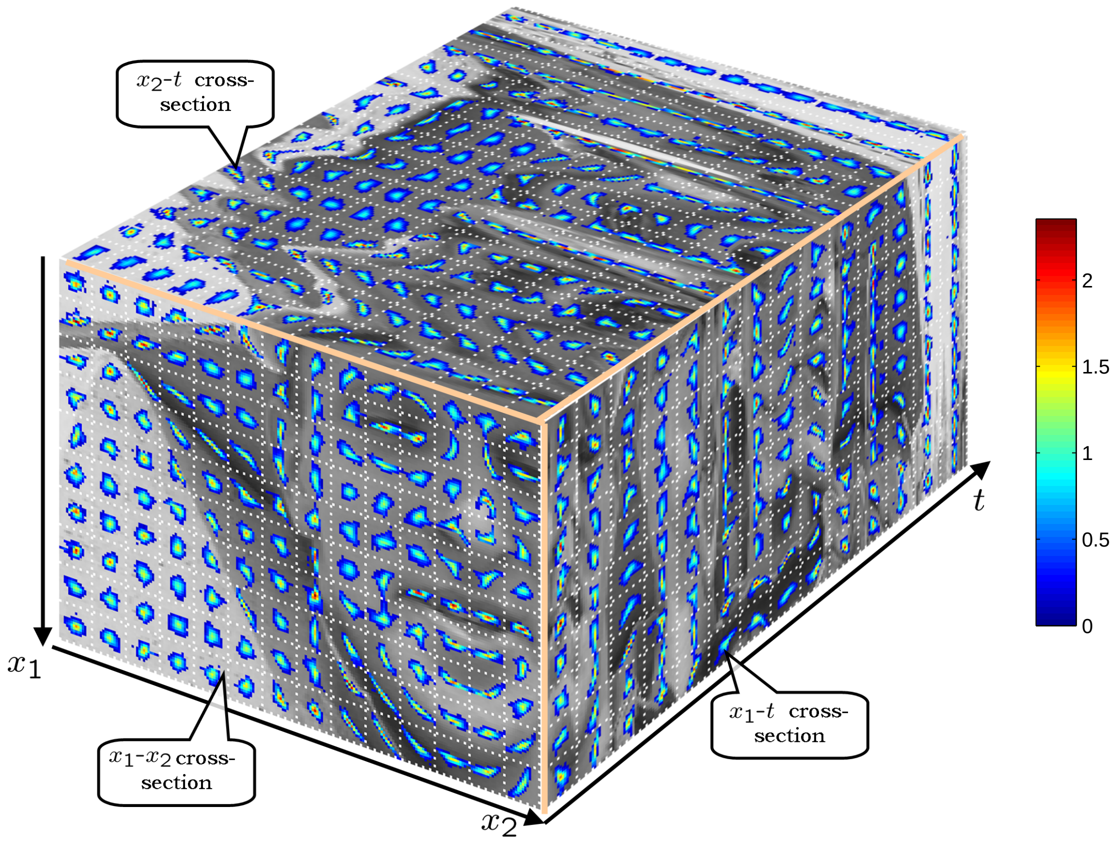


Figure 4.8: A graphical representation of 3-D steering kernel weights (4.31) for the Foreman sequence: The figure illustrate cross-sections of the steering kernel function $K_{\mathbf{H}_i}(\mathbf{x}_i - \mathbf{x})$ with (4.25) at every 15 pixels in horizontal, vertical, and time. For the illustration, we chose the analysis cubicle sizes $\omega = 15 \times 15 \times 15$ and $\xi_i = 5 \times 5 \times 5$.

brightness constancy equation (BCE) [91] in a local cubicle centered at $\mathbf{x}_i$, we can use the matrix of gradients $\mathbf{J}_i$ in (4.27) to write the BCE as

$$\mathbf{J}_i \widetilde{\mathbf{m}}_i = \mathbf{J}_i \begin{bmatrix} \mathbf{m}_i \\ 1 \end{bmatrix} = \underline{\mathbf{0}}. \tag{4.32}$$

Multiplying both sides of the BCE above by $\mathbf{J}_i^T$, we have

$$\mathbf{J}_i^T \mathbf{J}_i \widetilde{\mathbf{m}}_i = \mathbf{C}_i \widetilde{\mathbf{m}}_i \approx \underline{\mathbf{0}}. \tag{4.33}$$

Now invoking the decomposition of $\widehat{\mathbf{C}}_i$ in (4.28), we can write

$$\sum_{q=1}^{3} \varrho_q \mathbf{v}_q \left(\mathbf{v}_q^T \widetilde{\mathbf{m}}_i \right) \approx \underline{\mathbf{0}}. \tag{4.34}$$

The above decomposition shows explicitly the relationship between the motion vector and the principal orientation directions computed within the SKR framework. The most generic scenario in a small cubicle is one where the local texture features move with approximate uniformity. In this generic case, we have $\varrho_1, \varrho_2 \gg \varrho_3$, and it can be shown that the singular vector $\mathbf{v}_3$ (which we do not directly use) corresponding to the smallest singular value ϱ_3 can be approximately interpreted as the total least squares estimate of the homogeneous optical flow vector $\frac{\widetilde{\mathbf{m}}_i}{\|\widetilde{\mathbf{m}}_i\|}$ [92, 93]. As such, the steering kernel footprint will therefore spread along this direction, and consequently assign significantly higher weights to pixels along this implicitly given motion direction. In this sense, compensation for small local motions is taken care of implicitly by the assignment of the kernel weights. It is worth noting that a significant strength of using the proposed implicit framework (as opposed to the direct use of estimated motion vectors for compensation) is the flexibility it provides in terms of smoothly and adaptively changing the elongation parameters defined by the singular values in (4.29). This flexibility allows the accommodation of even complex motions, so long as their magnitudes are not excessively large. When the magnitude of the motions is large (relative to the support of the steering kernels, specifically) a basic form of coarse but explicit motion compensation will become necessary.

There are two approaches that we can consider to compensate for large displacement. In the MASK approach introduced in Section 4.2, we explicitly feed local motion vectors directly into 3-D kernels. More specifically, we construct 3-D kernels by shifting the 2-D (spatial) steering kernels by motion vectors. Moreover, in order to suppress artifacts in the estimated videos due to the errors in motion vectors, we compute the reliability of each local motion vector, and penalize the 2-D steering kernels accordingly. In the next section, we describe an alternative approach that does not require accurate motion vectors. In general, it is hard to estimate motions in the presence of occlusions and nonrigid transitions. As shown in Figure **4.8**, the 3-D steering kernel effectively fits them. Therefore, all we need is to compensate large displacements by shifting the video frames with whole pixel accuracy, and the 3-D steering kernels implicitly take the leftover motions into account as local 3-D image structures.

4.3.2 Kernel Regression with Rough Motion Compensation

Before formulating the 3-D SKR with motion compensation, first, let us discuss how the steering kernel behaves in the presence of relatively large motions[3]. In Figures **4.9**(a) and (b), we illustrate the contours of steering kernels for the pixel of interest located at the center of the middle frame. For the small displacement case illustrated in Figure **4.9**(a), the steering kernel ideally spreads across neighboring frames, taking advantage of information contained in the space-time neighborhood. Consequently, we can expect to see the effects of resolution enhancement and strong denoising. On the other hand, in the presence of large displacements as illustrated in Figure **4.9**(b), similar pixels, though close in the time dimension, are found far away in space. As a result, the estimated kernels will tend not to spread across time. That is to say, the net result is that the 3-D SKR estimates in effect default to the 2-D case. However, if we can roughly estimate the relatively large motion of the block and compensate (or

[3]It is important to note here that by large motions we mean speeds (in units of pixels per frame) which are larger than the typical support of the local steering kernel window, or the moving object's width along the motion trajectory. In the latter case, even when the motion speed is slow, we are likely to see temporal aliasing locally.

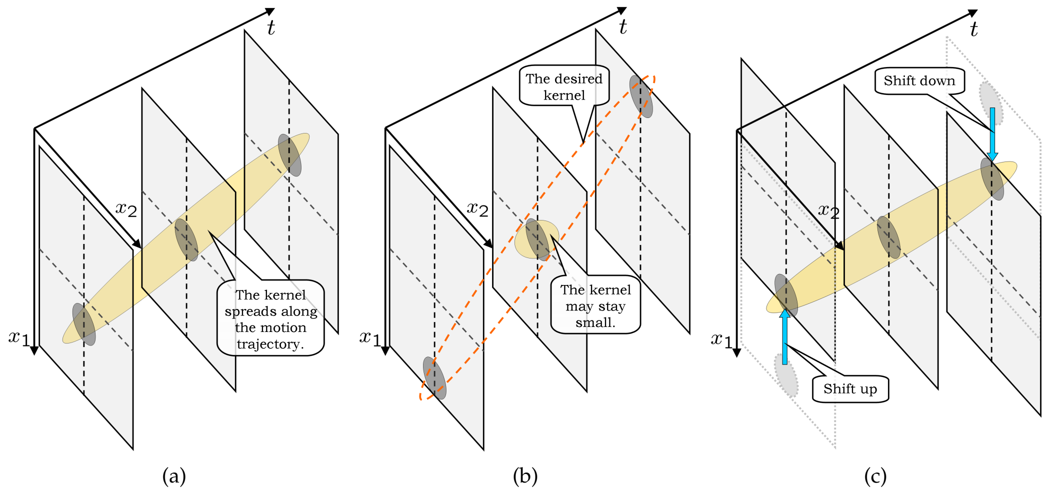
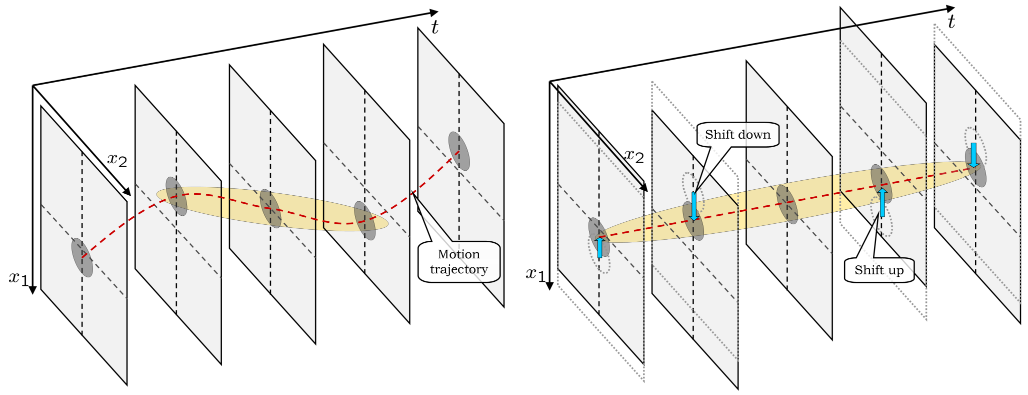

t
x_2
x_1
The kernel spreads along the motion trajectory.
(a)
The desired kernel
The kernel may stay small.
(b)
Shift down
Shift up
(c)

Figure 4.9: Steering kernel footprints for (a) a video with small displacements, (b) a video with large displacements, and (c) the video after neutralizing the large displacements.

"neutralize") for it, as illustrated in Figure **4.9**(c), and then compute the 3-D steering kernel, we find that it will again spread across neighboring frames and we regain the interpolation/denoising performance of 3-D SKR. The above approach can be useful even in the presence of aliasing when the motions are small but complex in nature. As illustrated in Figure **4.10**(b), if we cancel out these displacements, and make the motion trajectory smooth, the estimated steering kernel will again spread across neighboring frames and result in good performance.

In any event, it is quite important to note that the above compensation is done for the sole purpose of computing the more effective steering kernel weights. More specifically, (i) this "neutralization" of large displacements is *not* an explicit motion compensation in the classical sense invoked in coding or video processing, (ii) it requires absolutely no interpolation, and therefore introduces no artifacts, and (iii) it requires accuracy no better than a whole pixel.



(a) (b)

Figure 4.10: Steering kernel footprints for (a) a video with a complex motion trajectory, and (b) the video after neutralizing the relatively large displacements.

To be more explicit, 3-D SKR with motion compensation can be regarded as a two-tiered approach to handle a wide variety of transitions in video. Complicated transitions can be split into two different motion components: large whole-pixel motions ($\mathbf{m}_i^{\text{large}}$) and small but complex motion ($\mathbf{m}_i$):

$$\mathbf{m}_i^{\text{true}} = \mathbf{m}_i^{\text{large}} + \mathbf{m}_i, \tag{4.35}$$

where $\mathbf{m}_i^{\text{large}}$ is easily estimated by, for instance, optical flow or block matching algorithms, but $\mathbf{m}_i$ is much more difficult to estimate precisely.

Suppose a motion vector $\mathbf{m}_i^{\text{large}} = [m_{1i}^{\text{large}}, m_{2i}^{\text{large}}]^T$ is computed for each pixel in the video. We neutralize the motions of the given video data y_i by $\mathbf{m}_i^{\text{large}}$, to produce a new sequence of data $y(\tilde{\mathbf{x}}_i)$, as follows:

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + \begin{bmatrix} \mathbf{m}_i^{\text{large}} \\ 0 \end{bmatrix} (t_i - t), \tag{4.36}$$

where t is the time coordinate of interest. It is important to reiterate that since the motion estimates are rough (accurate to at best a single pixel) the formation of the sequence $y(\tilde{\mathbf{x}}_i)$ does not require any interpolation, and therefore no artifacts are introduced. Rewriting the 3-D SKR problem for the new sequence $y(\tilde{\mathbf{x}}_i)$, we have:

$$\min_{\{\boldsymbol{\beta}_n\}_{n=0}^{N}} \sum_{i=1}^{P} \left[y(\tilde{\mathbf{x}}_i) - \beta_0 - \boldsymbol{\beta}_1^T(\tilde{\mathbf{x}}_i - \mathbf{x}) - \boldsymbol{\beta}_2^T \text{vech}\left\{(\tilde{\mathbf{x}}_i - \mathbf{x})(\tilde{\mathbf{x}}_i - \mathbf{x})^T\right\} - \cdots \right]^2 K_{\widetilde{\mathbf{H}}_i}(\tilde{\mathbf{x}}_i - \mathbf{x}) \tag{4.37}$$

where the steering matrix $\widetilde{\mathbf{H}}_i$ is computed from the motion-compensated sequence $y(\tilde{\mathbf{x}}_i)$. Similar to the 2-D estimator (1.33), the above minimization yields the following pixel estimator at the position of interest ($\mathbf{x}$) as

$$\begin{aligned} \hat{z}(\mathbf{x}) &= \widehat{\beta}_0 = \mathbf{e}_1^T \left(\widetilde{\mathbf{X}}^T \widetilde{\mathbf{K}}^{\text{s}} \widetilde{\mathbf{X}}\right)^{-1} \widetilde{\mathbf{X}}^T \widetilde{\mathbf{K}}^{\text{s}} \, \tilde{\mathbf{y}} \\ &= \sum_{i=1}^{P} W_i(K, \widetilde{\mathbf{H}}_i, N, \tilde{\mathbf{x}}_i - \mathbf{x}) \, y(\tilde{\mathbf{x}}_i), \end{aligned} \tag{4.38}$$

where $\tilde{\mathbf{y}}$ is column-stacked vector of the given pixels ($y(\tilde{\mathbf{x}}_i)$), and $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{K}}^{\text{s}}$ are the basis matrix and the steering kernel weight matrix constructed with the motion compensated coordinates ($\tilde{\mathbf{x}}_i$); that is to say,

$$\tilde{\mathbf{y}} = \begin{bmatrix} y(\tilde{\mathbf{x}}_1), & y(\tilde{\mathbf{x}}_2), & \cdots, & y(\tilde{\mathbf{x}}_P) \end{bmatrix}^T, \quad \mathbf{b} = \begin{bmatrix} \beta_0, & \boldsymbol{\beta}_1^T, & \cdots, & \boldsymbol{\beta}_N^T \end{bmatrix}^T,$$

$$\widetilde{\mathbf{K}}^{\text{s}} = \text{diag}\begin{bmatrix} K_{\widetilde{\mathbf{H}}_1}(\tilde{\mathbf{x}}_1 - \mathbf{x}), & K_{\widetilde{\mathbf{H}}_2}(\tilde{\mathbf{x}}_2 - \mathbf{x}), & \cdots, & K_{\widetilde{\mathbf{H}}_P}(\tilde{\mathbf{x}}_P - \mathbf{x}) \end{bmatrix}, \tag{4.39}$$

and

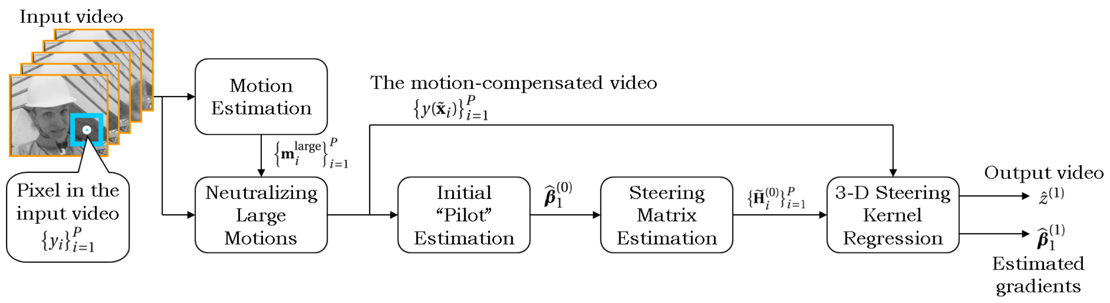
$$\widetilde{\mathbf{X}} = \begin{bmatrix} 1, & (\tilde{\mathbf{x}}_1 - \mathbf{x}), & \text{vech}\left\{(\tilde{\mathbf{x}}_1 - \mathbf{x})(\tilde{\mathbf{x}}_1 - \mathbf{x})^T\right\}, & \cdots \\ 1, & (\tilde{\mathbf{x}}_2 - \mathbf{x}), & \text{vech}\left\{(\tilde{\mathbf{x}}_2 - \mathbf{x})(\tilde{\mathbf{x}}_2 - \mathbf{x})^T\right\}, & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ 1, & (\tilde{\mathbf{x}}_P - \mathbf{x}), & \text{vech}\left\{(\tilde{\mathbf{x}}_P - \mathbf{x})(\tilde{\mathbf{x}}_P - \mathbf{x})^T\right\}, & \cdots \end{bmatrix}. \tag{4.40}$$

In the following section, we further elaborate on the implementation of the 3-D SKR for enhancement and super-resolution, including its iterative application.

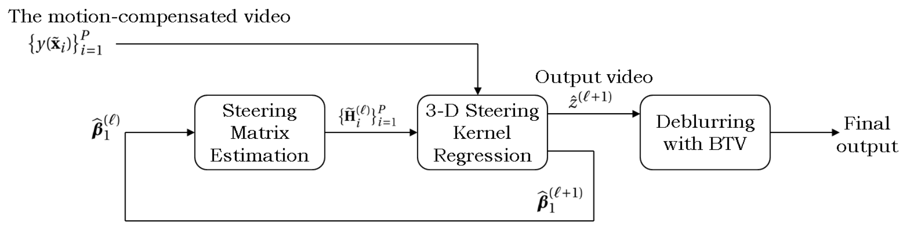
4.3.3 Implementation and Iterative Refinement

As we explained earlier, since the performance of the SKR depends on the accuracy of the orientations, we refine it to derive an iterative algorithm we call *iterative SKR* (ISKR), which results in improved orientation estimats and therefore a better final denoising and upscaling result. The extension for upscaling is done by first interpolating or upscaling using some reasonably effective low-complexity method (say the "classic" KR method) to yield what we call a *pilot* initial estimate. The orientation information is then estimated from this initial estimate and the SKR method is then applied to the input video data y_i which we embed in a higher resolution grid. To be more precise, the basic procedure, as shown in Figure **4.11** is as follow.

First, estimate the large motions ($\mathbf{m}_i^{\text{large}}$) of the given input sequence ($\{y_i\}_{i=1}^P$). Then using $\mathbf{m}_i^{\text{large}}$, we neutralize the large motions and generate a motion-compensated video sequence ($\{y(\tilde{\mathbf{x}}_i)\}_{i=1}^P$). Next, we compute the gradients ($\widehat{\boldsymbol{\beta}}_1^{(0)} = [\hat{z}_{x_1}(\cdot), \hat{z}_{x_2}(\cdot), \hat{z}_t(\cdot)]^T$) at the sampling positions $\{\tilde{\mathbf{x}}_i\}_{i=1}^P$ of the motion-compensated video. This process is indicated as the "pilot" estimate in the block diagram. Subsequently, we create steering matrices ($\widetilde{\mathbf{H}}_i^{(0)}$) for all the samples $y(\tilde{\mathbf{x}}_i)$ by (4.25) and (4.28). Once $\widetilde{\mathbf{H}}_i^{(0)}$ are available, we plug them into the kernel weight matrix (4.39) and estimate not only an unknown pixel value ($z(\mathbf{x})$) at a position of interest ($\mathbf{x}$) by (4.38) but also its gradients ($\widehat{\boldsymbol{\beta}}_1^{(1)}$). This is the initialization process shown in Figure **4.11**(a). Next, using $\widehat{\boldsymbol{\beta}}_1^{(1)}$, we re-create the steering matrices $\widetilde{\mathbf{H}}_i^{(1)}$. Since the estimated gradients $\widehat{\boldsymbol{\beta}}_1^{(1)}$ are also denoised and upscaled by SKR, the new steering matrices contain better orientation information. With $\widetilde{\mathbf{H}}_i^{(1)}$, we apply SKR to the embedded input video again. We repeat this process several times as shown in Figure **4.11**(b). While we do not discuss the convergence properties of this approach here, it is worth mentioning that typically, no more than a few iterations


Input video
Motion Estimation
The motion-compensated video
$\{y(\tilde{\mathbf{x}}_i)\}_{i=1}^P$
$\{\mathbf{m}_i^{\text{large}}\}_{i=1}^P$
Pixel in the input video $\{y_i\}_{i=1}^P$
Neutralizing Large Motions
Initial "Pilot" Estimation
$\widehat{\boldsymbol{\beta}}_1^{(0)}$
Steering Matrix Estimation
$\{\widehat{\mathbf{H}}_i^{(0)}\}_{i=1}^P$
3-D Steering Kernel Regression
Output video $\hat{z}^{(1)}$
$\widehat{\boldsymbol{\beta}}_1^{(1)}$
Estimated gradients

(a) Initialization


The motion-compensated video
$\{y(\tilde{\mathbf{x}}_i)\}_{i=1}^P$
$\widehat{\boldsymbol{\beta}}_1^{(\ell)}$
Steering Matrix Estimation
$\{\widehat{\mathbf{H}}_i^{(\ell)}\}_{i=1}^P$
3-D Steering Kernel Regression
Output video $\hat{z}^{(\ell+1)}$
Deblurring with BTV
Final output
$\widehat{\boldsymbol{\beta}}_1^{(\ell+1)}$

(b) Iteration

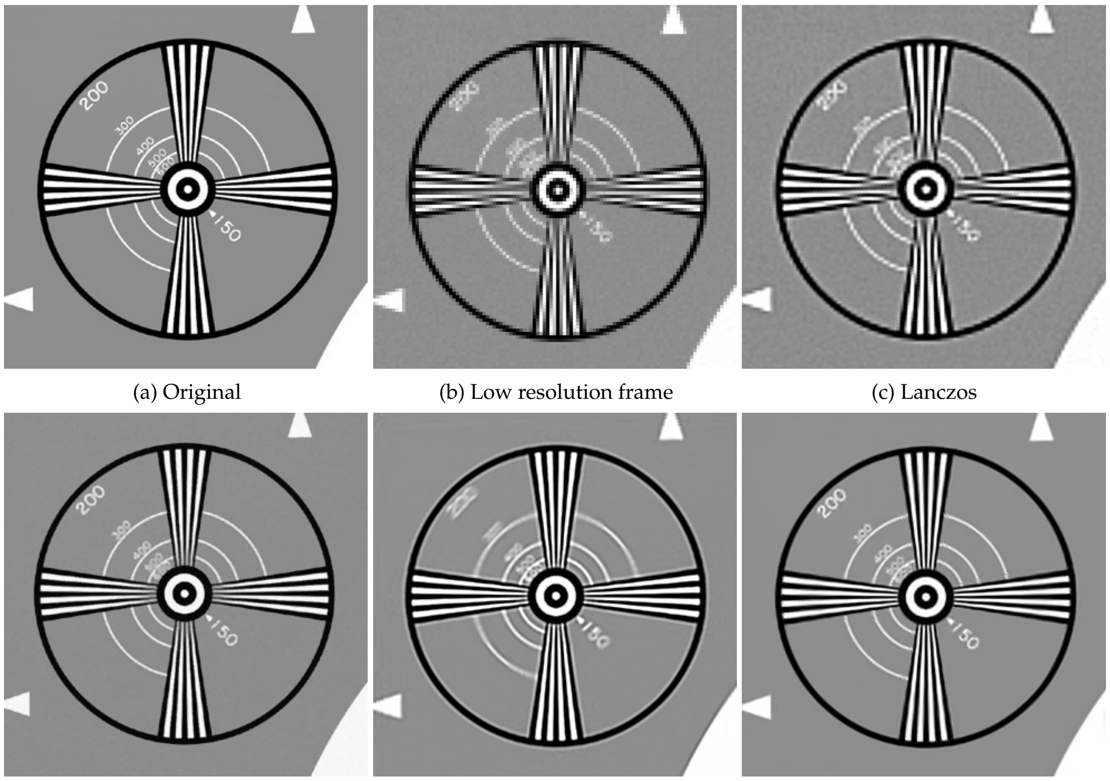
Figure 4.11: Block Diagram representation of the 3-D iterative steering kernel regression with motion compensation: (a) the initialization process, and (b) the iteration process.

are necessary to reach convergence[4]. Finally, we perform deblurring on the upscaled videos to recover the high frequency components. We use the deblurring with BTV (i.e. the deblurring estimator (3.3) with the BTV regularization (3.6)) in this work.

Figure **4.12** illustrates a simple super-resolution example, where we created 9 synthetic low resolution frames from the image shown in Figure **4.12**(a) by blurring with a 3×3 uniform PSF, shifting the blurred image by 0, 4, or 8 pixels[5] along the x_1- and x_2-axes, spatially downsampling with a factor $3 : 1$, and adding white Gaussian noise with standard deviation 2. One of the low resolution frames is shown in Fig-

[4]It is worth noting that the application of the iterative procedure results in a tradeoff of bias and variance in the resulting final estimate. As for an appropriate number of iterations, a relatively simple stopping criterion can be developed based on the behavior of the residuals (the difference images between the given noisy sequence and the estimated sequence) [94, 95].

[5]Note: this amount of shift creates severe temporal aliasing.



(a) Original (b) Low resolution frame (c) Lanczos

(d) Robust SR [2] (e) NLM base SR [80] (f) ISKR with motion comp.

Figure 4.12: A simple super-resolution example: (a) the original image, (b) one of 9 low resolution images generated by blurring with a 3×3 uniform PSF, spatially downsampling with a factor of $3:1$, and adding white Gaussian noise with standard deviation 2, (c) an upscaled image by Lanczos (single frame upscale), (d) an upscaled image by robust super-resolution (SR) [2], and (e) an upscaled image by non-local mean (NLM) based super-resolution [80], and (f) an upscaled image by 3-D ISKR with rough motion compensation. The corresponding PSNR values are (c)19.67, (d)30.21, (e)27.94, and (f)29.16[dB], respectively.

ure **4.12**(b). Then we created a synthetic input video by putting those low resolution images together in *random order*. Thus, the motion trajectory of the input video is not smooth and the 3-D steering kernel weights cannot spread effectively along time as illustrated in Figure **4.10**(a). The upscaled frames by Lanczos, robust super-resolution [2], non-local mean based super-resolution [80], and 3-D ISKR with rough motion estimation at time $t = 5$ are shown in Figures **4.12**(c)-(f), respectively.

In the presence of severe temporal aliasing arising from large motions, the task of accurate motion estimation becomes significantly harder. However, rough motion estimation and compensation is still possible. Indeed, once this compensation has taken place, the level of aliasing artifacts within the new data cubicle becomes mild, and as a result, the orientation estimation step is able to capture the true space-time orientation (and therefore implicitly the motion) quite well. This estimate then leads to the recovery of the missing pixel at the center of the cubicle, from the neighboring compensated pixels, resulting in true super-resolution reconstruction as shown in Figure **4.12**.

It is worth noting that while in the proposed algorithm in Figure **4.11**, we employ an SVD-based method for computing the 3-D orientations, other methods can also be employed such as that proposed by Farnebäck et al. using local tensors in [96]. Similarly, in our implementation, we used the optical flow framework [84] to compute the rough motion estimates. This step too can be replaced by other methods such as a block matching algorithm [97].

4.4 Video Upscaling Examples

The utility and novelty of the both algorithms, MASK and 3-D ISKR, lies in the fact that they are capable of both spatial and temporal (and therefore spatiotemporal) upscaling and super-resolution. Therefore, in this section, we study the performance of our method in both spatial and spatiotemporal cases.

Using two real sequences: Carphone and Texas video sequences, we first compare MASK and 3-D ISKR to non-local mean based super-resolution [80]. The algorithm proposed in [80] consists of multi-frame fusion with non-local mean based weighting [57], as well as explicit deblurring. For MASK and 3-D ISKR, we set the temporal window support to 5, and NL-based SR approach searches similar local patches across all the frame in time and and in a window of support 21×21 in space.

The first example shown in Figure **4.13** is a visual comparison of spatial upscaling and temporal frame interpolation results, using NLM-based SR [80], MASK, and 3-D ISKR. For this example, we used the Carphone video sequence in QCIF format (144×176 pixels, 30 frames) as input, and spatially upscaled the video by a factor of $1:3$. Figure **4.13**(a) shows the input frame at time $t = 27$ (upscaled by pixel-replication). The spatially upscaled results by single frame Lanczos interpolation, NL-based SR [80], MASK, and 3-D ISKR are shown in Figures **4.13**(b),(c),(f), and (h), respectively. In addition, Figures **4.13**(d)(f) and (g)(i) show spatiotemporally upscaled frames at $t = 26.5$ and 27.5 of Carphone sequence by MASK and 3-D ISKR, respectively. We used the same spatial upscaling factor of $1:3$ for the intermediate frames as well. Comparing to the result by Lanczos interpolation, all the adaptive methods, NLM-based SR, MASK, and 3-D ISKR, reconstruct high-quality upscaled frames, although each has a few artifacts: jaggy artifacts on edge regions for NLM-based SR and MASK, and overshoting artifact for 3-D ISKR.

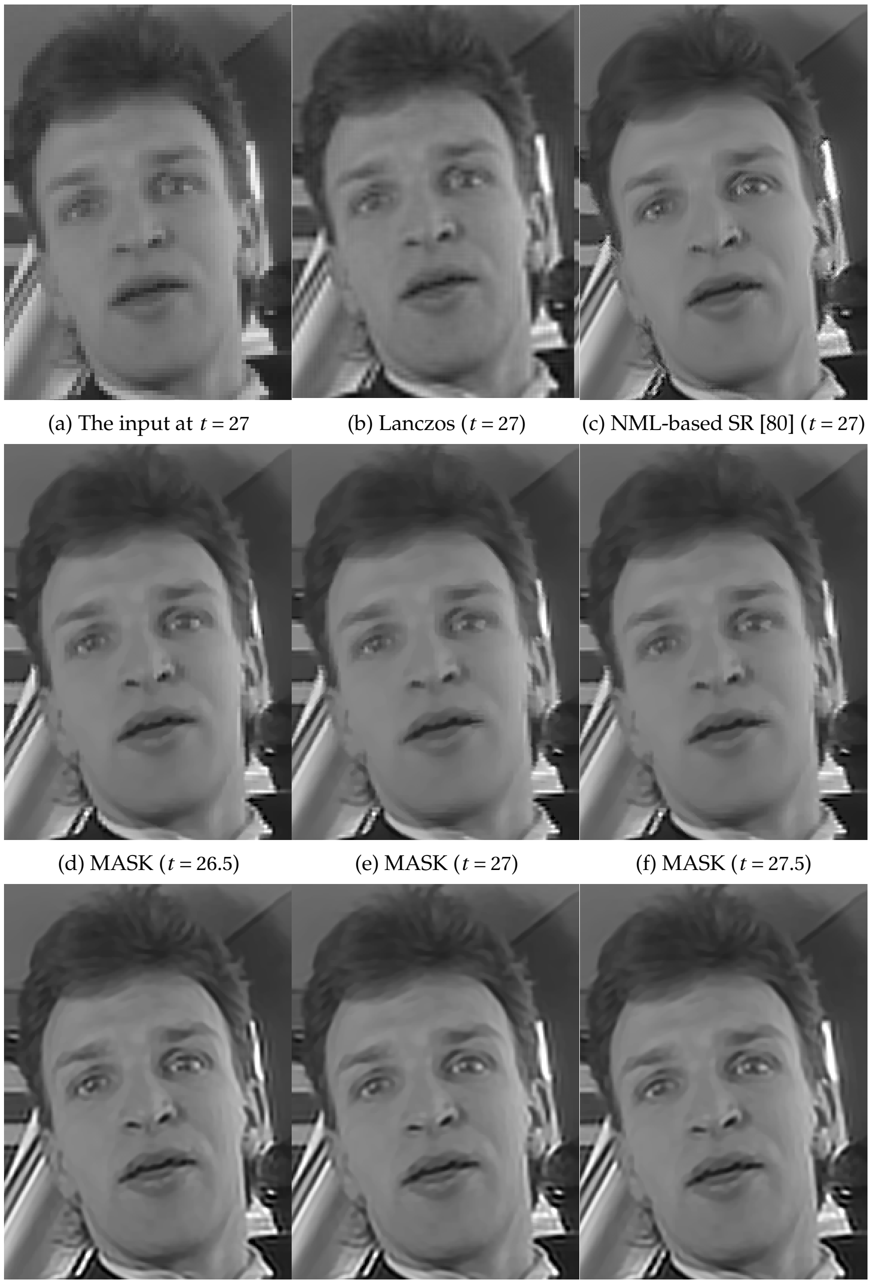
Next, similar to the Carphone example, the second example is also spatiotemporal video upscaling using Texas sequence (504×576 pixels, 30 frames). Figure **4.14**(a) shows an input frame of Texas sequence at time $t = 5$. Texas sequence also contains complicated motions, i.e., occlusion, 3-D rotation of human heads, and reflection on the helmet, and is degraded with compression artifacts (e.g. blocking). Video frames that were spatially upscaled with a factor of $1:3$ using NLM-based SR [80], 3-D ISKR, and MASK are shown in Figures **4.14**(b)-(d), respectively. In addition, Figures **4.14**(e)-

(h) show selected portions of the input frame, and the upscaled frames by NLM-based SR, 3-D ISKR, and MASK, respectively. Next, we estimated an intermediate frame at time $t = 5.5$ for the Texas sequence by 3-D ISKR and MASK, and the results are shown in Figure **4.15**. The intermediate frames are also spatially upscaled by the same factor $1:3$. Again, both 3-D ISKR and MASK produce high quality frames in which blocking artifacts are hardly visible while the important contents are preserved.

The final example is again spatiotemporal video upscaling. This time, we used Spin-Calender sequence (504×576 pixels, 30 frames). The sequence has relatively simpler motions comparing to Carphone and Texas sequences: the scene rotates clockwise with a constant speed. Using the sequence, we test MASK and 3-D ISKR and see how tolerant they are against motion errors. Although it is possible to find an accurate motion model for the underlying motions, we estimated the motions with the translational model. Hence, errors in the estimated motions are inescapable due to the violation of the motion model. Figure **4.16**(a) shows the input frame at time $t = 5$, and we estimated an intermediate frame at $t = 5.5$ with a spatial upscaling factor $1:2$. The estimated frames by 3-D ISKR and MASK are shown in Figures **4.16**(c) and (d), respectively, and Figures **4.16**(e)-(f) show the selected portions of (a)-(c), respectively. As seen in the figure, although both 3-D ISKR and MASK succeeded in suppressing compression noise, 3-D ISKR recovers finer details in the stripe region. This is because 3-D steering kernels adapt the residual motions that the translational motion estimation left behind, while the MASK approach simply penalizes the neighboring frames based on the motion errors. The temporal penalization of the MASK method can be regarded as a fail-safe mechanism for the motion estimation, that excludes the neighboring samples with unreliable motions from the local kernel regression in order not to produce artifacts. Of course, the more accurate the motions are, the MASK approach is also able to produce the higher quality frames.

Summary— In this chapter, we presented two different extensions of steering kernel regression for video upscaling. First, we proposed the MASK method that is capable of spatial upscaling with resolution enhancement, temporal frame interpolation, noise reduction, as well as sharpening. In the MASK approach, we construct 3-D kernels based on local motion vectors. The algorithm requires accurate motion estimation but doesn't use explicit motion compensation of video frames. Instead, the spatiotemporal kernel is oriented along the local motion trajectory, and subsequent kernel regression acts directly on the pixel data. Also, we reduce the computational cost of MASK by using a block-based motion model, using a quantized set of local orientations, and adapting the regression order. The adaptive regression order technique not only reduces the computational cost, but also provides sharpening while avoiding noise amplification.

Next, we proposed the 3-D SKR approach (a direct extension of 2-D SKR to 3-D) for spatiotemporal video upscaling. Traditionally, super-resolution reconstruction of image sequences has relied strongly on the availability of highly accurate motion estimations between frames. As is well-known, subpixel motion estimation is quite difficult, particularly in situations where the motions are complex in nature. As such, this has limited the applicability of many existing upscaling algorithms to simple scenarios. In this chapter, significantly, we illustrated that the need for explicit subpixel motion estimation can be avoided by the two-tiered approach presented in Section 4.3, which yields excellent results in both spatial and temporal upscaling.

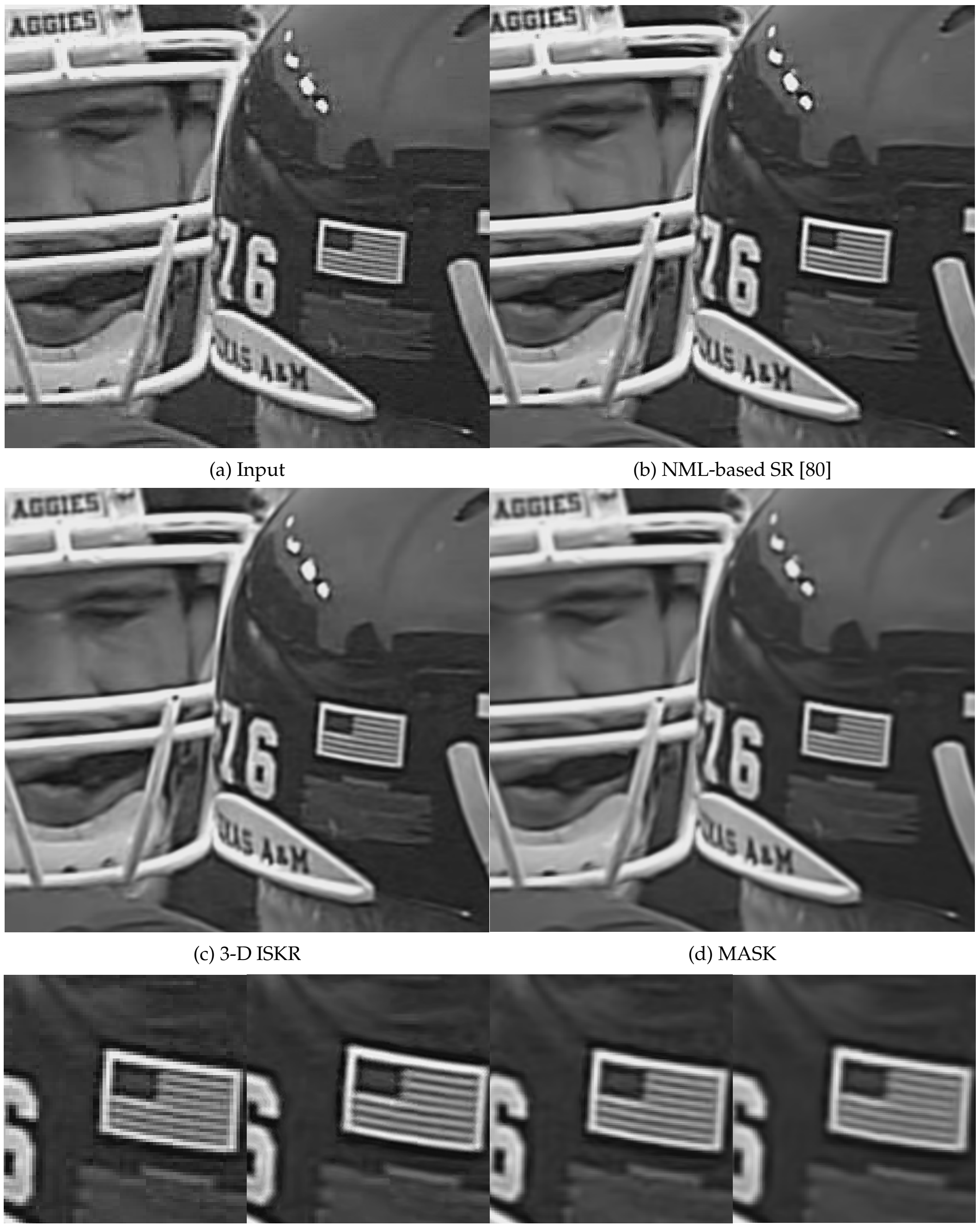


(a) The input at $t = 27$ (b) Lanczos ($t = 27$) (c) NML-based SR [80] ($t = 27$)

(d) MASK ($t = 26.5$) (e) MASK ($t = 27$) (f) MASK ($t = 27.5$)

(g) 3-D ISKR ($t = 26.5$) (h) 3-D ISKR ($t = 27$) (i) 3-D ISKR ($t = 27.5$)

Figure 4.13: A Carphone example of video upscaling with spatial upscaling factor 1 : 2: (a) the input video frame at time $t = 27$ (144×176, 30 frames), (b)-(c) upscaled frames by Lanczos interpolation and NLM-based SR method [80], respectively, (d)-(f) upscaled frames by MASK at $t = 26.5$, 27, and 27.5, respectively, and (g)-(i) upscaled frames by 3-D ISKR at $t = 26.5$, 27, and 27.5, respectively.

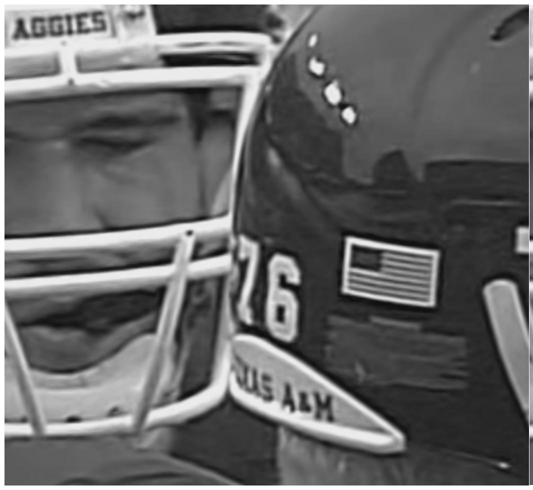


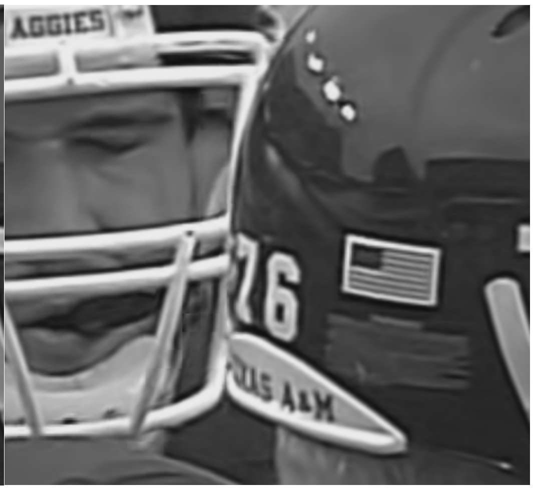
(a) Input (b) NML-based SR [80]

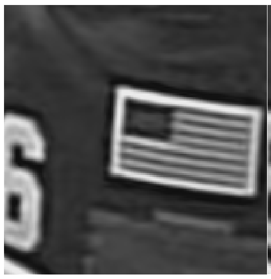
(c) 3-D ISKR (d) MASK

(e) Input (f) NML-based SR [80] (g) 3-D ISKR (h) MASK

Figure 4.14: Spatial upscaling of Texas video sequence: (a) the input frame at $t = 5$, (b)-(d) the upscaled video frames by NML-based SR [80], 3-D ISKR, and MASK, respectively. (e)-(h) Enlarged images of the input frame and the upscaled frames by NML-based SR, 3-D ISKR, and MASK, respectively.


(a) 3-D ISKR


(b) MASK


(c) 3-D ISKR

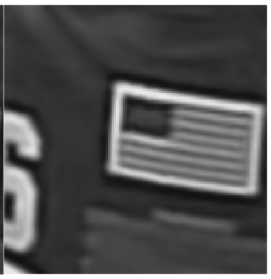
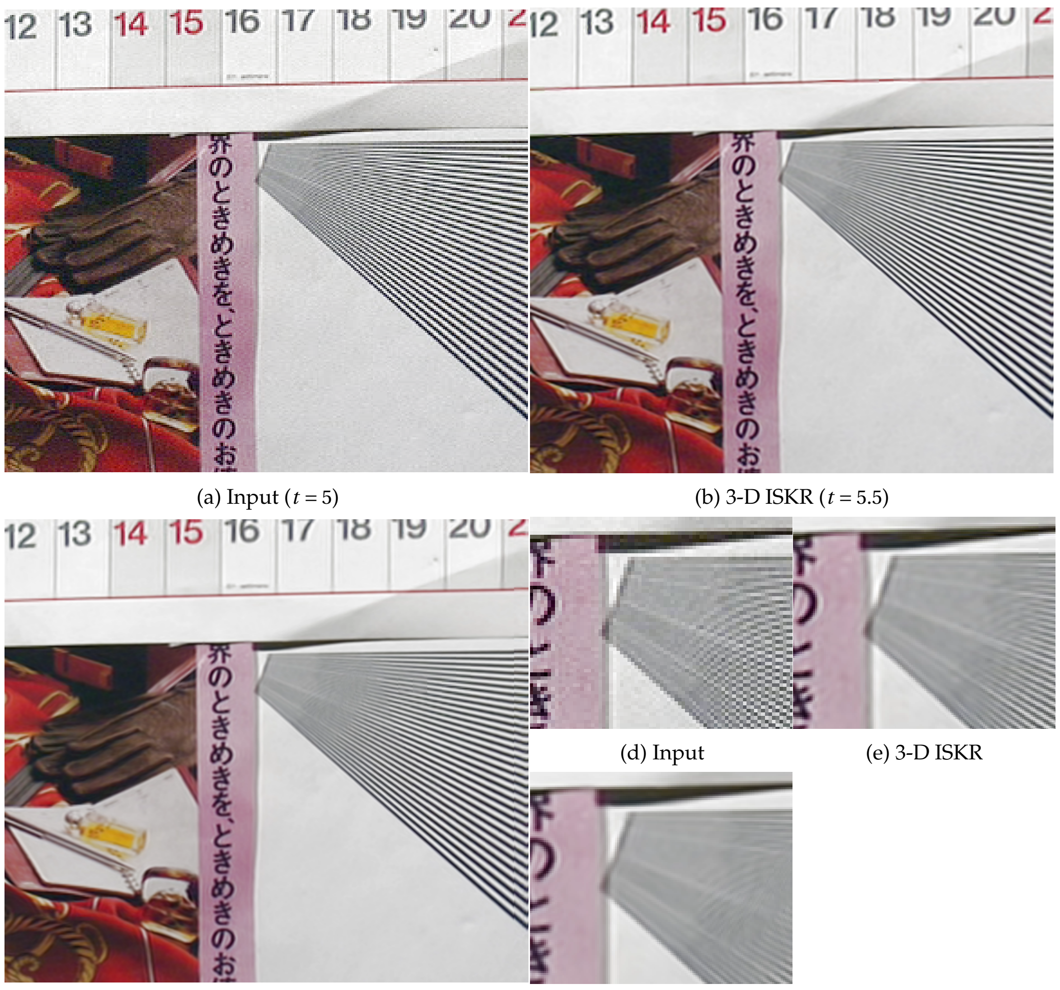

(d) MASK

Figure 4.15: Spatiotemporal upscaling of Texas video sequence: (a)-(b) the estimated intermediate frames at time $t = 5.5$ by 3-D ISKR and MASK, and (b)-(c) the enlarged images of the upscaled frames by 3-D ISKR and MASK, respectively.



(a) Input ($t = 5$)

(b) 3-D ISKR ($t = 5.5$)

(c) MASK ($t = 5.5$)

(d) Input

(e) 3-D ISKR

(f) MASK

Figure 4.16: Spatiotemporal upscaling of Spin Calendar video sequence: (a)-(c) the input frame at time $t = 5$, the estimated intermediate frames at time $t = 5.5$ by 3-D ISKR and MASK, and (d)-(f) the enlarged images of the input frame, the upscaled frames by 3-D ISKR and MASK, respectively.

Chapter 5

Video Deblurring

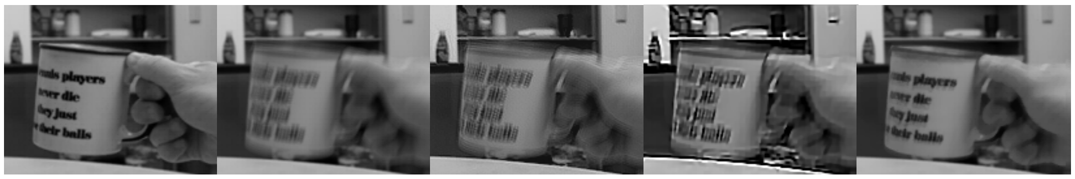
Abstract— In Chapter 4, we extended 2-D steering kernel regression into 3-D as an application of space-time video upscaling, but we did not address the blur effect. In general, video sequences are degraded by not only spatial blur, due to the camera aperture, but also *motion blur*, due to relative displacements between the camera and the objects. Although spatial deblurring is relatively well-understood by assuming that the blur kernel is shift-invariant, motion blur is not so when we attempt to deconvolve this motion blur on a frame-by-frame basis: this is because, in general, videos include complex, multi-layer transitions. Indeed, we face an exceedingly difficult problem in motion deblurring of a single frame when the scene contains motion occlusions. Instead of deblurring video frames individually, a fully 3-D deblurring method is examined in this chapter to reduce motion blur from a motion-blurred image sequence. The approach is free from knowledge of local motions. Most importantly, due to its inherent locally adaptive nature, the 3-D approach is capable of automatically deblurring the parts of the sequence which are motion blurred without segmentation, and without adversely affecting the rest of the spatiotemporal domain where such blur is not

present.

5.1 Introduction

The practical solutions to blind motion deblurring available so far largely only treat the case where the blur is a result of global motions due to the camera displacement [98, 99], rather than motion of the objects in the scene. When the motion blur is not global then the segmentation information is needed in order to identify what part of the image suffers from motion blur (typically due to the fast-moving objects). Consequently, the problem of deblurring moving objects in the scene is quite complex because it requres (i) segmentation of moving objects from the background, (ii) estimation of a spatial motion *point spread function* (PSF) for each moving object, (iii) deconvolution of the moving objects one-by-one with the corresponding PSFs, and finally (iv) putting the deblurred objects back together into a coherent and artifact free image or sequence [100, 101, 102, 103]. In order to perform the first two steps, i.e. segmentation and PSF estimation, one would need to carry out global/local motion estimation [104, 105, 106, 107]. Thus, the deblurring performance strongly depends on the accuracy of motion estimation and segmentation of moving objects. However, the errors in both are in general unavoidable, particularly in the presence of multiple motions, occlusions, or non-rigid motions, i.e. when there are any motions that violate parametric models or the standard optical flow brightness consistency constraint.

In this chapter, we present a motion deblurring approach for videos that is free of both motion estimation and segmentation. Briefly speaking, we point out and exploit what in hindsight seems obvious, through not exploited so far: that motion blur is by nature a temporal blur, which is caused by relative displacements of the camera



(a) the ground truth (b) a motion-blurred frame (c) Fergus *et al.* [98] (d) Shan *et al.* [99] (e) Proposed

Figure 5.1: An example of motion blur reduction: (a) the ground truth, (b) the blurred frame, (c)-(e) deblurred frames by Fergus's method [98], Shan's method [99], and the proposed 3-D deblurring approach.

and the objects in the scene while the camera shutter is opened. Therefore, a temporal blur degradation model is more appropriate and physically meaningful for the general motion delubrring problem than the usual spatial blur model. An important advantage of the use of the temporal blur model is that regardless of whether the motion blur is global (camera induced) or local (object induced) in nature, the temporal PSF stays shift-invariant whereas the spatial PSF must be considered shift-variant in essentially all state of the art frame-by-frame (or spatial) motion deblurring approaches [100, 101, 102, 103]. The example in Figure **5.1** illustrates the advantage of our aproach as compared to the spatial motion deblurring methods proposed by Fergus *et al.* [98] and Shan *et al.* [99]. Both methods are designed for the removal of the global blur effect cased by the camera displacement. The ground truth, a blurry frame, and the restored images by Fergus'methods, Shan's method, and our approach are shown in Figure **5.1**(a)-(e), respectively. As can be seen from this typical example, Fergus'and Shan's methods deblur the background, while in fact we wish to restore the details of the mug. We will discuss this example in more detail in Section 5.3. In the meantime, we briefly summarize some existing methods for motion deblurring problem next.

5.1.1 Existing Methods and a Path Ahead

Ben-Erza *et al.* [100], Tai *et al.* [101], and Cho *et al.* [102] proposed deblurring methods where the spatial motion PSF is obtained from the estimated motions. Ben-Erza *et al.* [100] and Tai *et al.* [101] use two different cameras: a low-speed high resolution camera and a high-speed low resolution camera, and capture two videos of the same scene at the same time. Then, they estimate motions using the high-speed low resolution video so that detailed local motion trajectories can be estimated, and the estimated local motions yield a spatial motion PSF for each moving object. On the other hand, again using two cameras, Cho *et al.* [102] take a pair of stereo images. The stereo images enable the separation of the images into the foreground and the background. The foreground and the background are often blurred with different PSFs. The separation is helpful in estimating the different PSFs individually, and the estimation process of the PSFs becomes more stable.

Whereas the deblurring methods in [100, 101, 102] obtain the spatial motion PSF based on the global/local motion information, Levin proposed a blind motion deblurring method using a relationship between the distribution of derivatives and the degree of blur [103]. With this in hand, the method estimates a spatial motion PSF for each segmented object. In order to speed up the PSF estimation process, the PSF is parameterized by two parameters (direction and length) as a 1-D box kernel.

Later, inspired by Levin's blind motion deblurring method, Fergus *et al.* [98] and Shan *et al.* [99] proposed blind deblurring methods for a single blurred image caused by a shaking camera. Although their methods are limited to global motion blur, using the relationship between the distribution of derivatives and the degree of blur proposed by Levin, they estimate a shift-invariant PSF without parametrization.

Ji *et al.* [108] and Dai *et al.* [109] also proposed derivative-based methods. Ji *et al.* estimate the spatial motion PSF by a spectral analysis of the image gradients, and Dai *et al.* obtain the PSF by studying how blurry the local edges are as indicated by local gradients. Recently, another blind motion deblurring method was proposed by Chen *et al.* [110] for the reduction of global motion-blur. They claim that the PSF estimation is more stable with two images of the same scene degraded by different PSFs, and also use a robust estimation technique to stabilize the PSF estimation process further.

All the methods mentioned above are similar in that they aim at removing motion blur by spatial (2-D) processing. In the presence of multiple motions, the existing methods would have to estimate shift-variant PSF or segment the blurred images by local motions (or depth maps). However, occlusions make the deblurring problem harder because pixel values around motion occlusions are a mixture of multiple objects moving in independent directions. In this chapter, we reduce the motion blur effect from videos by introducing the 3-D deblurring data model. Since the data model is more reflective of the actual data acquisition process, even under the presence of motion occlusions, the deblurring with 3-D blur kernel can remove both global and local motion blur effectively without segmentation or reliance on motion information.

Practically speaking, for videos, it is not always preferable to remove all the motion blur effect from video frames. Particularly, for videos with relatively low frame rate (e.g. $10-20$ frames per second), in order to show smooth trajectory of moving objects, motion blur (temporal blur) is often intentionally added. Thus, when removing (or more precisely "reducing") the motion blur from videos, we would need to increase the temporal resolution of the video. This operation can be thought of as the familiar frame-rate upconversion, with the following caveat: in our context, the intermediate frames are not the end results of interest, but as we will explain shortly, rather a means to obtain a deblurred sequence, at possibly the original frame-rate. It is worth noting that the temporal blur reduction is equivalent to shortening the exposure time of video frames. Typically, the exposure time τ_{e} is less than the time interval τ_{f} between the frames (i.e. $\tau_{\mathrm{e}} \leq \tau_{\mathrm{f}}$) as shown in Figure **5.2**(a). Many commercial cameras set τ_{e} to shorter than $0.5\tau_{\mathrm{f}}$ (see for instance [111]). Borissoff in [111] pointed out that τ_{e} should ideally depend on the speed of moving objects. Specifically, the exposure time should be half of the time that a moving objects runs through the scene width, or the temporal aliasing would be visible. In [112], Shechtman *et al.* presented a space-time super-resolution algorithm where *multiple* cameras capture the same scene at once with slight spatial and temporal displacements. Then, multiple low resolution videos in space and time are fused to obtain a spatiotemporally super-resolved video. As a post-processing step, they spatiotemporally deblur the super-resolved video so that the exposure time τ_{e} nearly equals to the frame interval τ_{f}. In the present chapter, we solve the more restricted problem of motion blur resotration from a single, possible low frame-rate, video sequence.

To sum up, (i) removing motion blur effects from video frames completely is not always desirable because it can produce temporal aliasing effects. Instead of removing, we reduce motion blur for each frame. (ii) Frame interpolation (also known as *frame rate upconversion*) is necessary in order to reduce temporal aliasing and to show smooth trajectories of moving objects. (iii) Unlike motion deblurring algorithms which address the problem in two dimensions [98, 99, 100, 101, 102, 108, 109, 110]. We spatiotemporally deblur videos with a shift-invariant 3-D PSF, which is effective for any kind of motion blur. To obtain the 3-D PSF, we simply need the exposure time τ_{e} of the input videos (with is generally available from the camera setting) and the desired τ_e and τ_f of the output videos.

This chapter is organized as follows: in Section 5.2, we present the data model in 3-D and extend the image (2-D) deblurring technique with total variation (TV) regularization into 3-D for the video deblurring problem. The 3-D TV approach is effective in suppressing both noise and ringing effects. In Section 5.3, we show some motion deblurring examples of videos.

5.2 Video Deblurring in 3-D

In this section, we extend the single image (2-D) deblurring technique with TV regularization space-time (3-D) motion deblurring for videos. Ringing suppression is of importance because the ringing effect in time creates significant visual distortion for the output videos.

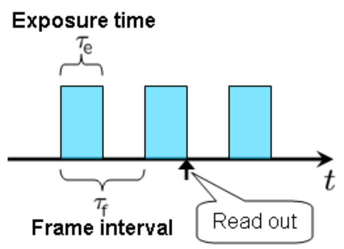
5.2.1 The Data Model

The exposure time τ_e of videos taken with a standard camera is always shorter than the frame interval τ_f as illustrated in Figure **5.2**(a). It is generally impossible to reduce motion blur by temporal deblurring when $\tau_\mathrm{e} < \tau_\mathrm{f}$ (i.e. the temporal support of the PSF is shorter than the frame interval τ_f). This is because the standard camera captures one frame at a time (the camera reads a frame out of the image sensor and resets the sensor). Unlike the spatial sampling rate, the temporal sampling rate is always below the Nyquist rate. This is an electromechanical limitation of the standard video camera. One way to have a high speed video with $\tau_\mathrm{e} > \tau_\mathrm{f}$ is to fuse multiple videos captured by multiple cameras at the same time with slight time delay as shown in Figure **5.2**(b).

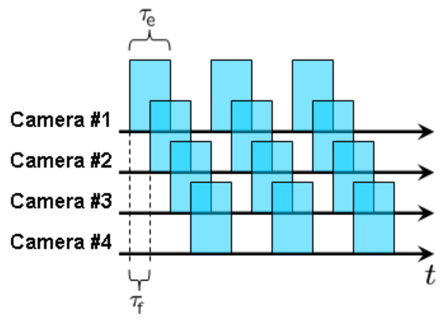
As we mentioned earlier, the technique is referred to as *space-time super-resolution* [112] or *high speed videography* [113]. After the fusion of multiple videos into a high speed video, the frame interval becomes shorter than the exposure time and we can carry out the temporal deblurring to reduce the motion blur effect.

An alternative to using multiple-cameras is to generate intermediate frames, which may be obtained by frame interpolation (e.g. [114], or our proposed method, 3-D SKR and MASK, in Chapter 4), so that the new frame interval $\tilde{\tau}_{\mathrm{f}}$ is now smaller than τ_{e} as illustrated in Figure **5.2**(c). Once we have the video sequence with $\tau_{\mathrm{e}} > \tilde{\tau}_{\mathrm{f}}$, the temporal deblurring reduces τ_{e} to be nearly equal to $\tilde{\tau}_{\mathrm{f}}$, and the video shown in Figure **5.2**(d) is our desired output. It is worth noting that, in the most general setting, generation/interpolation of temporally intermediate frames is indeed a very challenging problem. However, since our interest lies mainly in the removal of motion blur, the temporal interpolation problem is not quite as complex as the general setting. In the most general case, the space-time super-resolution method [112] employing multiple cameras may be the only practical solution. Of course, it is possible to apply the frame interpolation for the space-time super-resolved video to generate an even higher speed video. However, in this chapter, we focus on the case where only a single video is available and show that out intermediate frame interpolation, 3-D SKR, enables the motion deblurring.

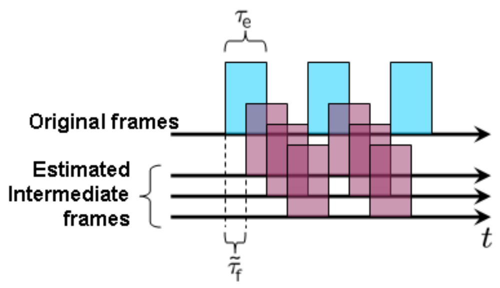
Figure **5.3** illustrates an idealized forward model which we adopt in this paper. More precisely, the camera captures the first frame by temporally integrating the first few frames (say the first, second and third frames) of the desired video $\mathbf{u}$, and the second frame by integrating, for example, the fifth frame and the following two



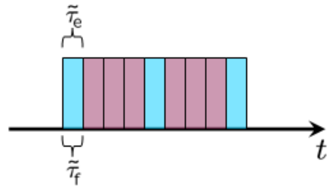
(a) Standard camera



(b) Multiple cameras [112]



(c) Frame-rate upconversion



(d) Temporally deblurred

Figure 5.2: A schematic representation of the exposure time τ_e and the frame interval τ_f: (a) a standard camera, (b) multiple videos taken by multiple cameras with slight time delay is fused to produce a high frame rate video, (c) the original frames with estimated intermediate frames, and (d) the output frames temporally deblurred.

frames[1]. Next, the frames are spatially downsampled due to the limited number of pixels on the image sensor. We can regard spatial and temporal sampling mechanisms of the camera altogether as space-time downsampling effect, as shown in Figure **5.3**.

In this work, we estimate the desired output **u** by a two-step approach: (i) space-time upscaling and (ii) space-time deblurring. In Chapter 4, we proposed a space-time upscaling method, where we left the motion (temporal) blur effect untreated, and removed only the out-of-focus spatial blur with a shift-invariant (2-D) PSF with TV regularization. In this chapter, we study the reduction of the spatial and tem-

[1]This frame accumulation represents the temporal blur effect, and the skips of the temporal sampling position can be regarded as temporal downsampling

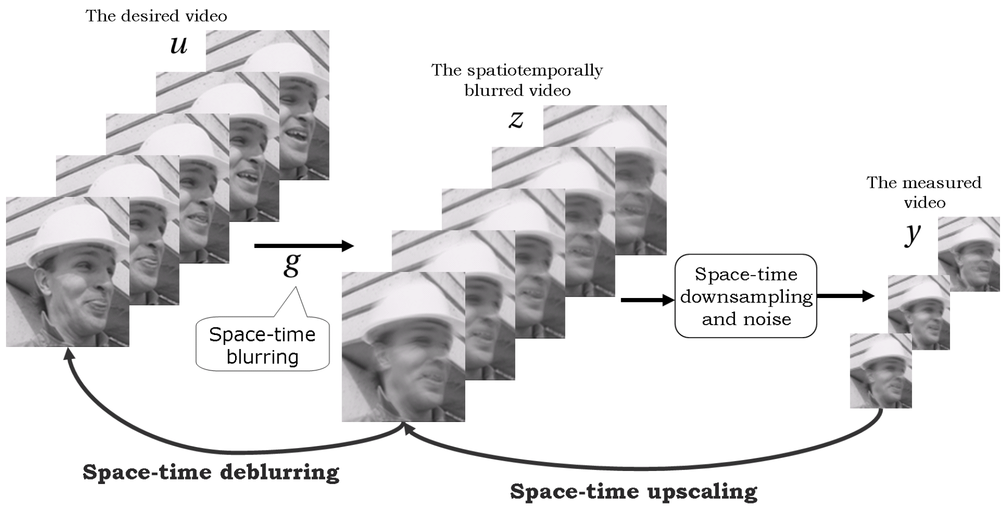


Figure 5.3: The forward model addressed in this paper, and we estimate the desired video **u** by two-step approach: (i) space-time upscaling, and (ii) space-time deblurring.

poral blur effects simultaneously with a shift-invariant (3-D) PSF. A 3-D PSF is effective because the out-of-focus blur and the temporal blur (frame accumulation) are both shift-invariant. PSF becomes shift-variant when we convert the 3-D PSF into 2-D temporal slices which yield the spatial PSF due to the moving objects for frame-by-frame deblurring. Again, unlike the existing methods [98, 99, 100, 101, 102, 108, 109, 110], after the space-time upscaling, no motion estimation or scene segmentation is required for the space-time deblurring.

Having graphically introduced our data model in Figure **5.3**, we define the mathematical model between the blurred data denoted y and the desired signal u with a 3-D PSF g as:

$$y(\mathbf{x}) = z(\mathbf{x}) \downarrow + \varepsilon = (g * u)(\mathbf{x}) \downarrow + \varepsilon_i, \tag{5.1}$$

where ε_i is the noise, $\mathbf{x} = [x_1, x_2, t]$, $\downarrow$ is the downsampling operator, $*$ is the convolution operator, and g is the combination of spatial blur g_s and the temporal blur g_τ

$$g(\mathbf{x}) = g_s(x_1, x_2) * g_\tau(t). \tag{5.2}$$

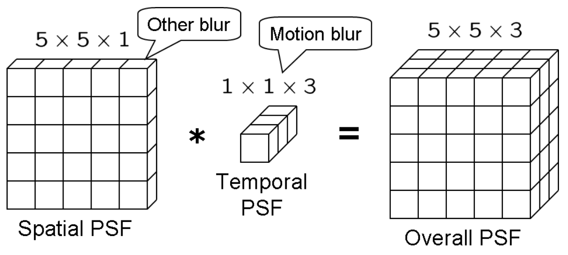


Figure 5.4: The overall PSF kernel in video (3-D) is given by the convolution of the spatial and temporal PSF kernels.

In this paper, we find u by a two-step approach by breaking the data model into

$$\text{Spatiotemporal upsampling model} \quad : \quad y(\mathbf{x}) = z(\mathbf{x}) \downarrow + \varepsilon, \tag{5.3}$$

$$\text{Spatiotemporal deblurring model} \quad : \quad z(\mathbf{x}) = (g * u)(\mathbf{x}). \tag{5.4}$$

In matrix form, if the sizes of the spatial and temporal PSF kernels are $N \times N \times 1$ and $1 \times 1 \times \tau$, respectively, then the overall PSF kernel has size $N \times N \times \tau$ as illustrated in Figure **5.4**. Since any unknown pixel value is coupled with its space-time neighbors due to the space-time blurring operation, it is preferable that we rewrite the data models (5.3) and (5.4) in matrix form as

$$\underline{\mathbf{y}} = \mathbf{D}\underline{\mathbf{z}} + \underline{\boldsymbol{\varepsilon}}, \tag{5.5}$$

$$\underline{\mathbf{z}} = \mathbf{G}\underline{\mathbf{u}}, \tag{5.6}$$

respectively, where $\mathbf{y} \in \mathcal{R}^{\frac{L}{r_s} \times \frac{M}{r_s} \times \frac{T}{r_t}}$ is the blurred noise-ridden low resolution video ($\tau_e > \tau_f$) with spatial and temporal downsampling factors r_s and r_t, $\mathbf{D} \in \mathcal{R}^{\frac{L}{r_s}\frac{M}{r_s}\frac{T}{r_t} \times LMT}$ represents the downsampling operation, $\mathbf{z} \in \mathcal{R}^{L \times M \times T}$ is the blurred version of the high resolution video that is available after the space-time video upscaling by the 3-D SKR method (4.37), $\mathbf{u} \in \mathcal{R}^{L \times M \times T}$ is the video of interest, $\mathbf{G} \in \mathcal{R}^{LMT \times LMT}$ represents the blurring operation, and $\boldsymbol{\varepsilon} \in \mathcal{R}^{L \times M \times T}$ is noise. The matrices with underscore represent that they are lexicographically ordered into column-stack vector form (e.g. $\underline{\mathbf{z}} \in \mathcal{R}^{LMT \times 1}$).

5.2.1.1 Space-Time (3-D) Deblurring

Assuming that, at the space-time upscaling stage, noise is effectively suppressed by 3-D SKR, the important issue that we need to carefully treat in the deblurring stage is the suppression of the ringing artifacts, particularly, across time. The ringing effect in time may cause undesirable flicker when we play the output video. Therefore, the deblurring approach should smooth the output pixel across not only space but also time. To this end, we propose a 3-D deblurring method with the 3-D version of total variation to recover the pixels across space and time:

$$\underline{\hat{\mathbf{u}}} = \arg\min_{\underline{\mathbf{u}}} \left\{ \left\| \underline{\mathbf{z}} - \mathbf{G}\underline{\mathbf{u}} \right\|_2^2 + \lambda \left\| \boldsymbol{\Gamma}\underline{\mathbf{u}} \right\|_1 \right\}, \tag{5.7}$$

where λ is the regularization parameter, and $\boldsymbol{\Gamma}$ is a high-pass filter. Specifically, we implement the TV regularization as

$$\left\| \boldsymbol{\Gamma}\underline{\mathbf{u}} \right\|_1 \Rightarrow \sum_{l=-1}^{1} \sum_{m=-1}^{1} \sum_{t=-1}^{1} \left\| \underline{\mathbf{u}} - \mathbf{S}_{x_1}^{l} \mathbf{S}_{x_2}^{m} \mathbf{S}_{\mathrm{t}}^{t} \underline{\mathbf{u}} \right\|_1 \tag{5.8}$$

where $\mathbf{S}_{x_1}^{l}$, $\mathbf{S}_{x_2}^{m}$, and $\mathbf{S}_{\mathrm{t}}^{t}$ are the shift operators that shift the video $\underline{\mathbf{u}}$ toward x_1, x_2, and t-directions with l, m, and t-pixels, respectively. We iteratively minimize the cost $C(\underline{\mathbf{u}}) = \left\| \underline{\mathbf{z}} - \mathbf{G}\underline{\mathbf{u}} \right\|_2^2 + \lambda \left\| \boldsymbol{\Gamma}\underline{\mathbf{u}} \right\|_1$ in (5.7) with (5.8) to find the deblurred sequence $\underline{\hat{\mathbf{u}}}$ using the steepest descent method:

$$\underline{\hat{\mathbf{u}}}^{(\ell+1)} = \underline{\hat{\mathbf{u}}}^{(\ell)} + \mu \left. \frac{\partial C(\underline{\mathbf{u}})}{\partial \underline{\mathbf{u}}} \right|_{\underline{\mathbf{u}} = \underline{\hat{\mathbf{u}}}^{(\ell)}} \tag{5.9}$$

where μ is the step size and

$$\frac{\partial C(\underline{\mathbf{u}})}{\partial \underline{\mathbf{u}}} = -\mathbf{G}^T \left(\underline{\mathbf{z}} - \mathbf{G}\underline{\mathbf{u}} \right) + \lambda \sum_{l=-1}^{1} \sum_{m=-1}^{1} \sum_{t=-1}^{1} \left(\mathbf{I} - \mathbf{S}_{x_1}^{-l} \mathbf{S}_{x_2}^{-m} \mathbf{S}_{\mathrm{t}}^{-t} \right) \operatorname{sign}\left(\underline{\mathbf{u}} - \mathbf{S}_{x_1}^{l} \mathbf{S}_{x_2}^{m} \mathbf{S}_{\mathrm{t}}^{t} \underline{\mathbf{u}} \right). \tag{5.10}$$

5.3 Experiments

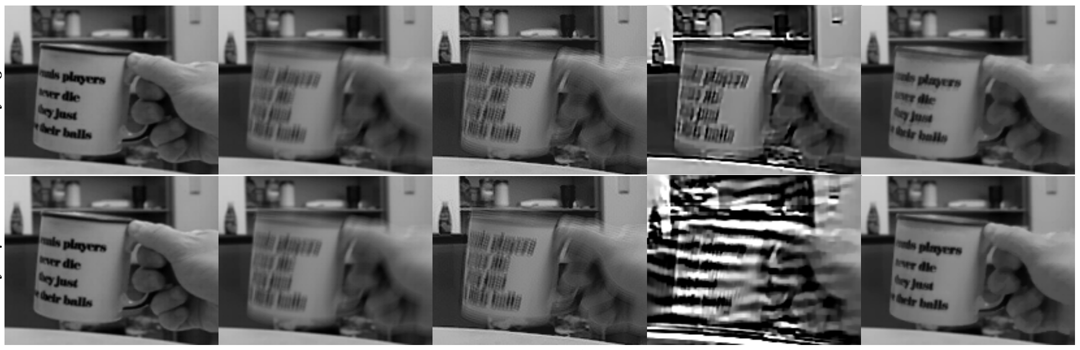
We illustrate the performance of our proposed technique on both real and simulated sequences. To begin, we first illustrate motion deblurring performance on two sequences, namely, the Cup sequence and ToyRobo sequence, with simulated motion blur[2]. The Cup example is the one we briefly showed in the introduction. This sequence contains relatively simple transitions, i.e. the cup moves upward. On the other hand, in the ToyRobo sequence, a toy robot walks to the right, and therefore it yields more complicated transitions. Figures. **5.5**(a) and **5.6**(a) show the ground truth frames of each sequence, and Figures. **5.5**(b) and **5.6**(b) show the motion blurred frames generated by taking average of 5 consecutive frames, i.e. the corresponding PSF in 3-D is $1 \times 1 \times 5$ uniform. The deblurred images of Cup and ToyRobo sequences by Fergus' method [98], Shan's method[3] [99] and our approach (5.7) with $(\mu, \lambda) = (0.75, 0.04)$ for Cup sequence and $(\mu, \lambda) = (0.75, 0.04)$ for ToyRobo sequence are shown in Figures. **5.5**(c)-(e) and **5.6**(c)-(e), respectively. Figures. **5.5**(f)-(j) and Figures. **5.6**(f)-(j) show the selected regions of the video frames Figures. **5.5**(a)-(e) and Figures. **5.6**(a)-(e) at time $t = 6$, respectively. The corresponding PSNR[4] and SSIM[5] values are indicated in the figure captions. Also, Figure. **5.7** shows the plots of the PSNR and SSIM values of the iterative deblurring approach with 3-D TV (5.9) for the Cup and ToyRobo frames at $t = 6$. It is worth noting here again that, although motion occlusions are present

[2]In order to examine how well the motion blur will be removed, we do not take the out-of-focus spatial blur into account for all the experiments.

[3]The software is available at `http://w1.cse.cuhk.edu.hk/ ~leojia/programs/deblurring/deblurring.htm`. We set the parameter "`noiseStr`" to 0.05 and used the default setting for the other parameters for all the examples.

[4]Peak Signal to Noise Ratio = $10 \log_{10}\left(\frac{255^2}{\text{Mean Square Error}}\right)$ [dB].

[5]The software for Structure SIMilarity index is available at `http://www.ece.uwaterloo.ca/~z70wang/research/ssim/`.


(a) The ground truth (b) Blurred frames (c) Fergus *et al.* [98] (d) Shan *et al.* [99] (e) Proposed (5.7)

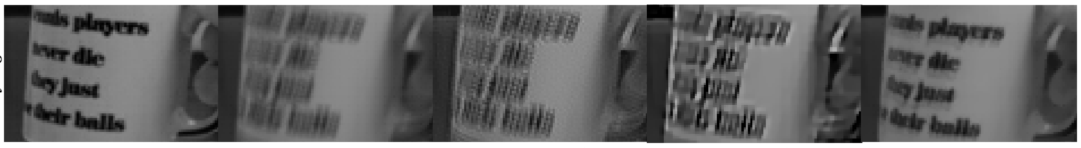

(f) The ground truth (g) Blurred frames (h) Fergus *et al.* [98] (i) Shan *et al.* [99] (j) Proposed (5.7)

Figure 5.5: A motion (temporal) deblurring example of the Cup sequence (130×165, 16 frames) in which a cup moves upward: (a) 2 frames of the ground truth at times $t = 6$ to 7, (b) the blurred video frames generated by taking the average of 5 consecutive frames (the corresponding temporal PSF is $1 \times 1 \times 5$ uniform) (PSNR[dB]: 23.76(top), 23.68(bottom), and SSIM: 0.76(top), 0.75(bottom)), (c)-(e) the deblurred frames by Fergus's method [98] (PSNR[dB]: 22.58(top), 22.44(bottom), and SSIM: 0.69(top), 0.68(bottom)), Shan's method [99] (PSNR[dB]: 18.51(top), 10.75(bottom), and SSIM: 0.57(top), 0.16(bottom)), the proposed 3-D TV method (5.7) (PSNR[dB]: 32.57(top), 31.55(bottom), and SSIM: 0.98(top), 0.97(bottom)), respectively. The figures (f)-(j) are the selected regions of the video frames (a)-(e) at time $t = 6$, respectively.

in the both sequences, the 3-D deblurring requires neither segmentation nor motion estimation.

The next experiment shown in Figure. **5.8** is a real example, where we deblur a low temporal resolution sequence degraded by real motion blur. The sequence consists of 5 frames[6], and the second and third frames are shown in Figure. **5.8**(a). Motion blur can be seen in the foreground (i.e. texts), and there is no blur in the background. Similar to the previous experiments, we first deblurred those frames individually by Fergus' and Shan's methods. Their deblurred results are in Figure. **5.8**(c) and (d), respectively.

[6]The full video sequence is available online at `http://videoprocessing.ucsd.edu/~stanleychan/research/SPL_2010.html`.

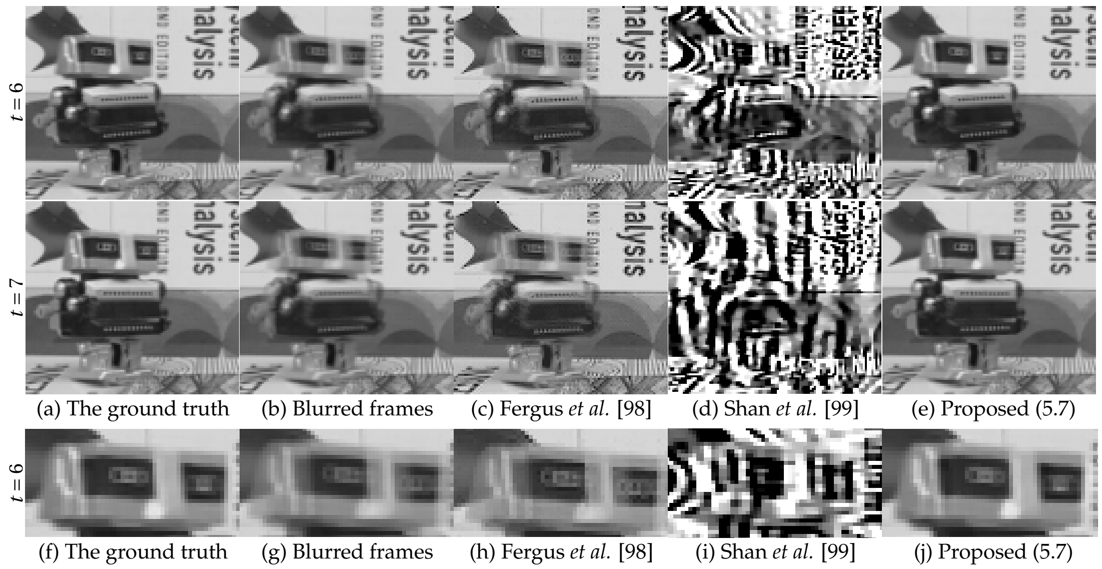
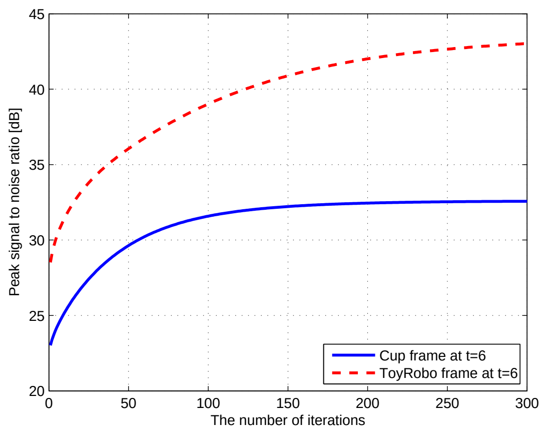


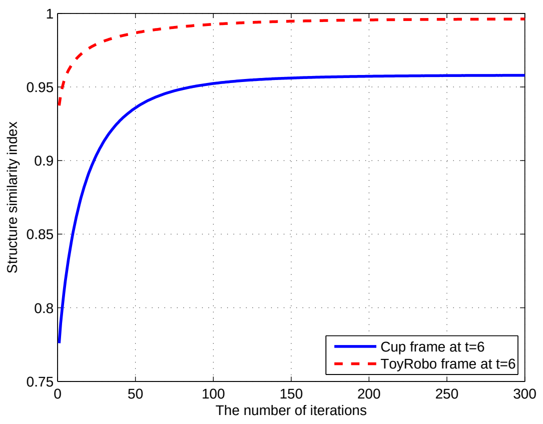
Figure 5.6: A motion (temporal) deblurring example of the ToyRobo sequence (84×124, 16 frames) in which a cup moves upward: (a) 2 frames of the ground truth at times $t = 6$ to 7, (b) the blurred video frames generated by taking the average of 5 consecutive frames (the corresponding PSF is $1 \times 1 \times 5$ uniform) (PSNR[dB] : 27.83(top), 24.62(bottom), and SSIM : 0.93(top), 0.87(bottom)), (c)-(e) the deblurred frames by Fergus's method [98] (PSNR[dB] : 25.71(top), 24.84(bottom), and SSIM : 0.90(top), 0.88(bottom)), Shan's method [99] (PSNR[dB] : 10.88(top), 8.69(bottom), and SSIM : 0.26(top), 0.14(bottom)), the proposed 3-D TV method (5.7) (PSNR[dB] : 43.02(top), 39.61(bottom), and SSIM : 0.99(top), 0.99(bottom)), respectively. The figures (f)-(j) are the selected regions of the video frames (a)-(e) at time $t = 6$, respectively.

For our method, temporal upscaling is necessary before deblurring. Here it is indeed the case that exposure time is shorter than the frame interval ($\tau_e < \tau_f$) as shown in Figure. **5.2**(a). Using the 3-D SKR method (4.37), we upscaled the sequence with the temporal upscaling factor[7] $1:8$ in order to generate intermediate frames to have the sequence as illustrated in Figure. **5.2**(c). Then we deblurred the upscaled video with a $1 \times 1 \times 8$ uniform PSF by the 3-D TV method (5.7) with $(\mu, \lambda) = (0.75, 0.06)$. The selected deblurred frames are shown in Figure. **5.8**(d), and selected regions of the input and

[7]Since the magnitude of the motion speed around texts is about 8 pixels per frame, we chose the temporal upscaling factor $1:8$ in order to reduce the motion blur completely.



(a) Peak signal to noise ratio (PSNR)



(b) Structure similarity index (SSIM)

Figure 5.7: Plots of (a) PSNR and (b) SSIM values of the iterative deblurring method with the proposed 3-D TV method (5.9) for the frames of Cup and ToyRobo sequences at $t = 6$. For the iterations, we set the step size μ in (5.9) 0.75.

deblurred frames at $t = 2$ are shown for comparison in Figure. **5.9**.

The last example is another real example. This time we used the Foreman sequence in CIF format. Figure **5.10**(a) shows one frame of the cropped input sequence (170×230, 10 frames) at time $t = 6$. In this example, we upscaled the Foreman sequence using 3-D SKR (4.37) with spatial and temporal upscaling factor of $1:2$ and $1:8$, respectively, and Figure **5.10**(e) show the estimated intermediate frame at time $t = 5.5$ and the estimated frame at $t = 6$. These frames are the intermediate results of our two-step deblurring approach). The 3-D SKR method successfully estimated the blurred intermediate frames, as seen in the figures, and the motion blur is spatially variant; the man's face is blurred as a result of the out-of-plane rotation of his head. In this time, we deblur the upscaled frames using Fergus' and Shan's methods [98, 99], and the proposed 3-D deblurring method. The deblurred frames are in Figures. **5.10**(b)-(d), respectively, and Figures. **5.10**(f)-(i) and (j)-(n) are the selected regions of the frames shown in (a)-(e) at $t = 5.5$ and 6, respectively. In addition, in order to compare the per-

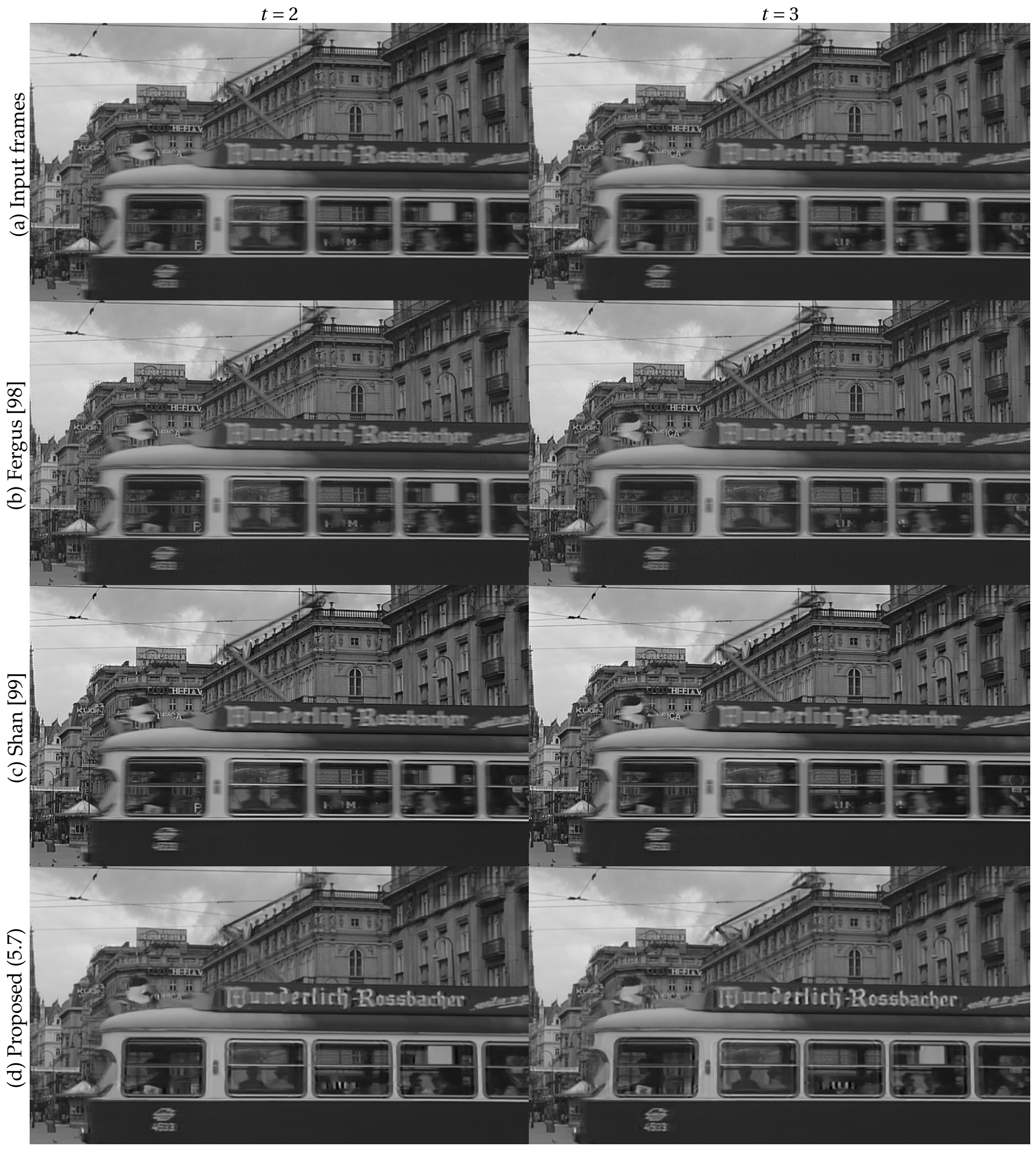
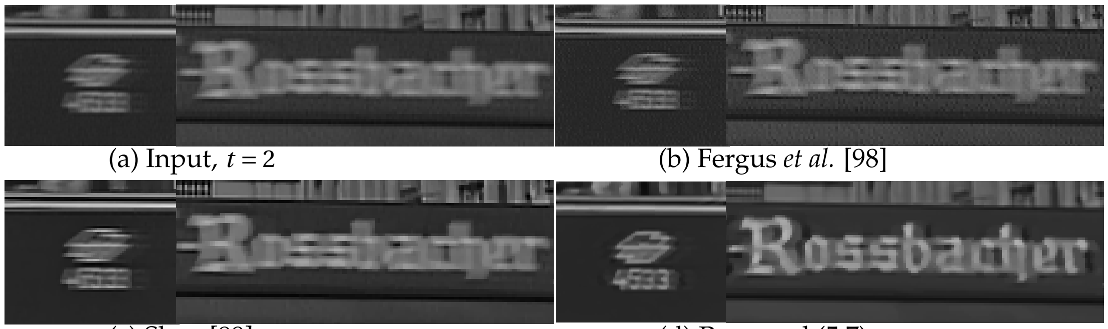


Figure 5.8: A motion (temporal) deblurring example of the StreetCar sequence (120×290, 5 frames) with real motion blur: (a) 2 frames of the ground truth at times $t = 2$ to 3, (b)-(c) the deblurred frames by Fergus's method [98], Shan's method [99], and (d) the deblurred frames by the proposed 3-D TV method (5.7) using a $1 \times 1 \times 8$ uniform PSF.

formance of our proposed method to Fergus' and Shan's methods, in Figure **5.11**, we compute the absolute residuals (the absolute difference between the deblurred frames



(a) Input, $t = 2$ (b) Fergus *et al.* [98]

(c) Shan [99] (d) Proposed (5.7)

Figure 5.9: Selected regions from the frame at $t = 2$ of the StreetCar sequence: (a) the input frame, (b)-(d) the deblurred results by Fergus *et al.* [98], Shan *et al.* [99], and the proposed 3-D TV (5.7) method, respectively.

shown in Figures. **5.10**(b)-(d) and the estimated frames shown in Figures. **5.10**(e) in this case). The results illustrate that our 3-D deblurring approach successfully recovers more details of the scene, such as the man's eye pupils, the outlines of the face and nose even without scene segmentation.

Summary— In this chapter, instead of removing the motion blur as spatial blur, we proposed deblurring with a 3-D space-time invariant PSF. The results showed that we could avoid segmenting video frames based on the local motions, and the temporal deblurring effectively removed motion blur even in the presence of motion occlusions.

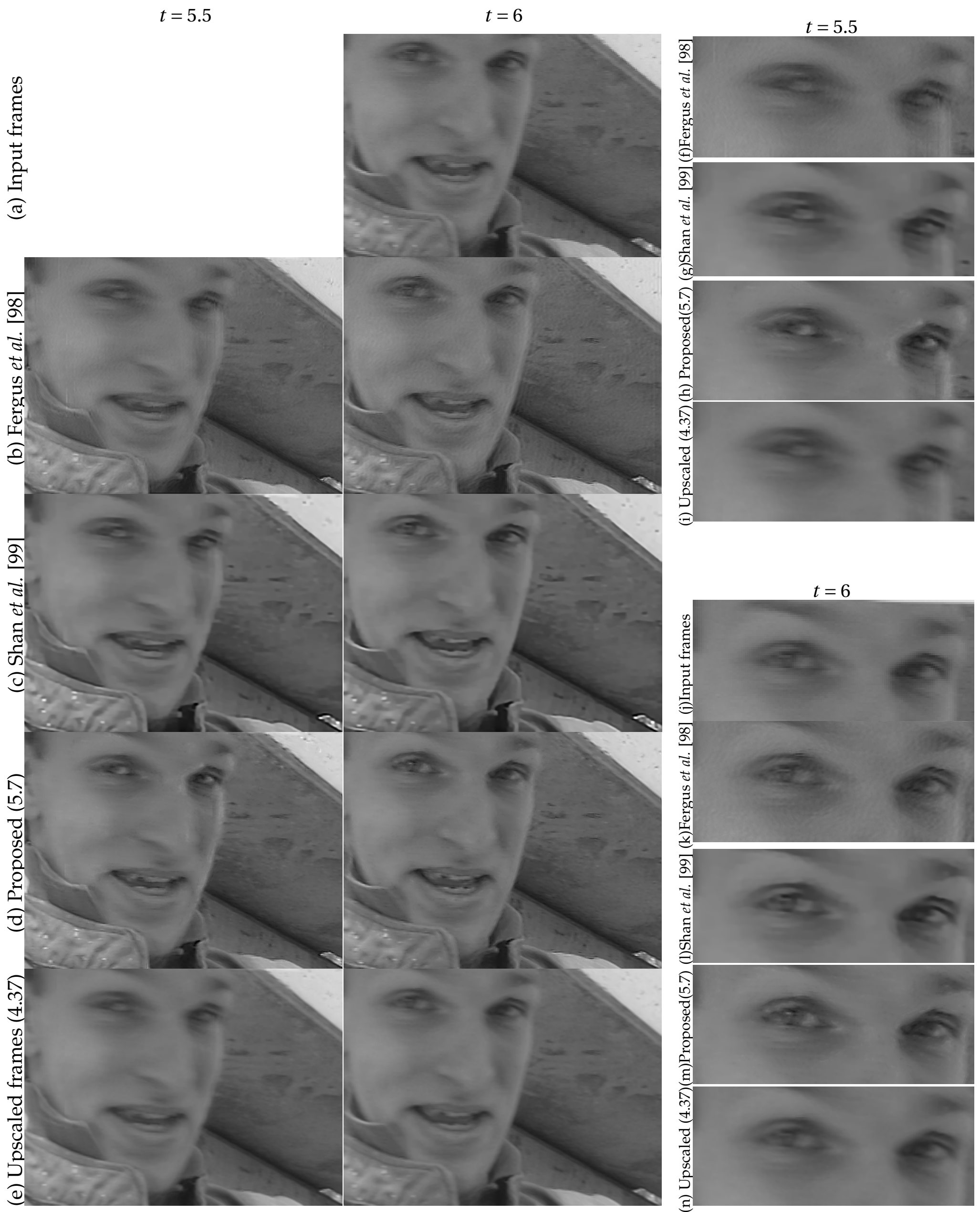
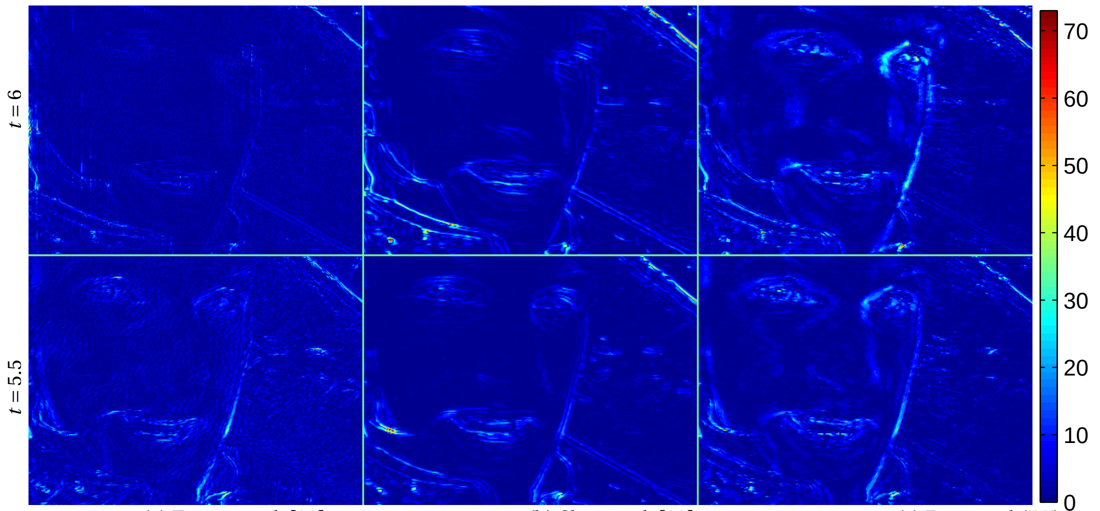


Figure 5.10: A 3-D (spatio-temporal) deblurring example of the Foreman sequence in CIF format: (a) the cropped frame at time $t = 6$, (b)-(c) the deblurred results of the upscaled frame shown in (e) by Fergus' method [98], Shan's method [99], (d) the deblurred frames by the proposed 3-D TV method (5.7) using a $2 \times 2 \times 2$ uniform PSF, and (e) the upscaled frames by 3-D SKR (4.37) at time $t = 6$ and 6.5 in both space and time with the spatial and temporal upscaling factors of $1:2$ and $1:8$, respectively. The figures (f)-(i) and (j)-(n) are the selected regions of the frames shown in (a)-(e) at $t = 6$ and 6.5.



(a) Fergus *et al.* [98] (b) Shan *et al.* [99] (c) Proposed (5.7)

Figure 5.11: Deblurring performance comparisons using absolute residuals (the absolute difference between the deblurred frames shown in Figures. **5.10**(b)-(d) and the estimated frames shown in Figures. **5.10**(e)): (a) Fergus' method [98], (b) Shan's method [99], and our proposed method (5.7).

Chapter 6

Conclusion and Future Works

6.1 Conclusion

In this thesis, we studied a non-parametric point estimation approach, *kernel regression* (KR), and proposed a data-adaptive extension, *steering kernel regression* (SKR), which locally learns the optimal filter coefficients from the underlying multi-dimensional data. We applied the proposed approach to a wide variety of problems in image/video processing, such as denoising, interpolation, deblurring, and super-resolution. The experimental results of both simulated and real data showed that the proposed method was competitive to state of the art methods.

▷ **Chapter 1** — We reviewed the basic concept of a non-parametric approach called *kernel regression* (KR) for image processing. The advantage of the non-parametric approach is that it is capable of finding missing pixels and smoothing noise-ridden pixels no matter how the pixels are spaced out. Hence, KR is applicable to a wide variety of tasks, including, denoising, upscaling, interpolation, deblurring, and super-resolution. However, the classic kernel regression estimates a pixel value by a spatially adaptive (but linear) combination of its nearby pixels.

- ▷ **Chapter 2** — In order to improve the performance of the kernel regression, we proposed a data-adaptive alternative to the classic kernel regression approach, where we learn the optimal filter coefficients from not only the spatial distances between the pixel of interest and its neighbors but also the radiometric distances (differences in pixel intensity). The resulting approach smooths pixels along the local orientation structure. In this chapter, we generalized the bilateral filter [7, 8] by the bilateral kernel regression (BKR), and proposed the steering kernel regression (SKR) and its iterative implementation which improves the filtering performance further. We showed the effectiveness of the proposed iterative SKR for denoising, interpolation, and multi-frame superresolution tasks comparing to the state of the art method in each task.

- ▷ **Chapter 3** — The blurring effect is one of the significant degradation factors in imaging systems. In this chapter, using the data model in matrix form involving the blurring operator, we proposed a regularized deblurring estimator, where we derived both the likelihood term and regularization term based on the kernel regression approach. Using the steering kernel weights, the proposed deblurring estimator implicitly sharpens parts of the image where there is any structure so that it suppresses both noise and ringing effects effectively. The experimental results show that it is superior to state of the art deblurring methods. Furthermore, we demonstrated that the proposed method is still applicable to upscaling by replacing the deblurring operator by the downsampling operator.

- ▷ **Chapter 4** — We extended the 2-D data-adaptive method into two different 3-D methods. The 3-D methods enhance the resolution of video in both space and time. The first extension is the motion-assisted steering kernel (MASK) method where we explicitly estimate local motion trajectories and construct a set of 2-D (spatial) steering kernels along the trajectories to obtain 3-D kernels (MASK). Since the MASK method relies on motions, its performance strongly depends on the accuracy of the motion estimation. To reduce the negative impact from the motion errors, we compute the reliability of the local motions and penalize the neighboring frames. Moreover, in order to reduce the computational load, we used a look-up table to obtain the steering matrix of each pixel without singular value decomposition, and proposed an adaptive regression order method. It automatically set the regression order to zero in flat areas and compute the second order kernel weights in closed form in textured areas.

The other approach, 3-D SKR, is a direct extension of 2-D SKR. The advantage of the 3-D SKR method is that it relies on 3-D local orientation information. The 3-D orientation is a combination of spatial and temporal local orientations, where the temporal orientation can be regarded as motion trajectories. Thus, we can avoid motion estimation to subpixel accuracy, although rough motion compensation is unavoidable when the motion is large. Furthermore, similar to the 2-D SKR approach, we implement this 3-D SKR approach with an iterative scheme to improve its performance.

Both MASK and 3-D SKR work well for simulated and real videos, and their performance are competitive to the existing video super-resolution methods. In addition, our proposed methods, MASK and 3-D SKR, are capable of increasing the temporal resolution as well as the spatial resolution unlike the existing methods which are limited to either spatial or temporal upscaling.

▷ **Chapter 5** — Motion blur is another electromechanical limitation of the camera which significantly degrades the visual quality of videos. In Chapter 4, although we ignored the motion blur effect, the temporal upscaling capability of our 3-D approach enables us to reduce the motion blur without motion estimation or scene segmentation, unlike the existing motion deblurring methods.

First, we reviewed the existing motion deblurring methods in the literature, and found that most of them deal with the motion blur as a spatial blur. A practical approach is to estimate motions and then construct a 2-D motion blur kernel based on the estimated motions. When there are multiple motions in the scene, image segmentation is necessary to deblur segment-by-segment with different 2-D motion blur kernels. Another practical approach is the blind deblurring method where the 2-D motion blur kernel and the deblurred image are estimated at the same time. However the blind approach is limited to the case where the blur is uniform across the entire region.

Although many existing motion deblurring methods are spatial in nature, the motion blur is, in fact, a temporal shift-invariant blur. After temporally upscaling the input video by 3-D SKR, we can reduce the motion blur by deblurring with a shift-invariant 3-D blur kernel. Therefore, neither motion estimation nor scene segmentation is necessary at the deblurring stage.

Table 6.1: Choices of possible kernel functions

Name	Kernel Functions
Epanechnikov	$K_h(\mathbf{x}_i - \mathbf{x}) = \begin{cases} \frac{3}{4}\left(1 - \frac{\lVert\mathbf{x}_i-\mathbf{x}\rVert_2^2}{h^2}\right) & \text{for } \frac{\lVert\mathbf{x}_i-\mathbf{x}\rVert_2}{h} < 1 \\ 0 & \text{otherwise} \end{cases}$
Biweight	$K_h(\mathbf{x}_i - \mathbf{x}) = \begin{cases} \frac{15}{16}\left(1 - \frac{\lVert\mathbf{x}_i-\mathbf{x}\rVert_2^2}{h^2}\right)^2 & \text{for } \frac{\lVert\mathbf{x}_i-\mathbf{x}\rVert_2}{h} < 1 \\ 0 & \text{otherwise} \end{cases}$
Triangle	$K_h(\mathbf{x}_i - \mathbf{x}) = \begin{cases} 1 - \frac{\lVert\mathbf{x}_i-\mathbf{x}\rVert_2}{h} & \text{for } \frac{\lVert\mathbf{x}_i-\mathbf{x}\rVert_2}{h} < 1 \\ 0 & \text{otherwise} \end{cases}$
Laplacian	$K_h(\mathbf{x}_i - \mathbf{x}) = \frac{1}{2h}\exp\left(-\frac{\lVert\mathbf{x}_i-\mathbf{x}\rVert_2}{h}\right)$

6.2 Future Directions

The future directions we discuss below are mainly categorized into (i) how to improve the visual quality of the outputs, (ii) how to reduce the computational complexity, and (iii) other problems to which our proposed approach can be applied.

6.2.1 The Choice of Kernel Function

In kernel density estimation [11], there are several choices for the kernel function $K(\mathbf{x}_i - \mathbf{x})$, other than Gaussian kernel function which we used in this work. For image processing, not only improving the numerical performance, but also the visual quality of the output images are of importance. Table **6.1** shows some possible choices of the kernel function, and Figure **6.1** illustrates the possible kernel functions in 1-D comparing to Gaussian kernel function. Figures **6.2** and **6.3** show the visual comparison of different kernel functions for image upscaling. In this examples, we downsampled the original Lena (512×512) and Parrot (256×256) images with a factor of 2:1 without applying any filter or adding noise. Figures **6.2**(a) and **6.3**(a) show the selected

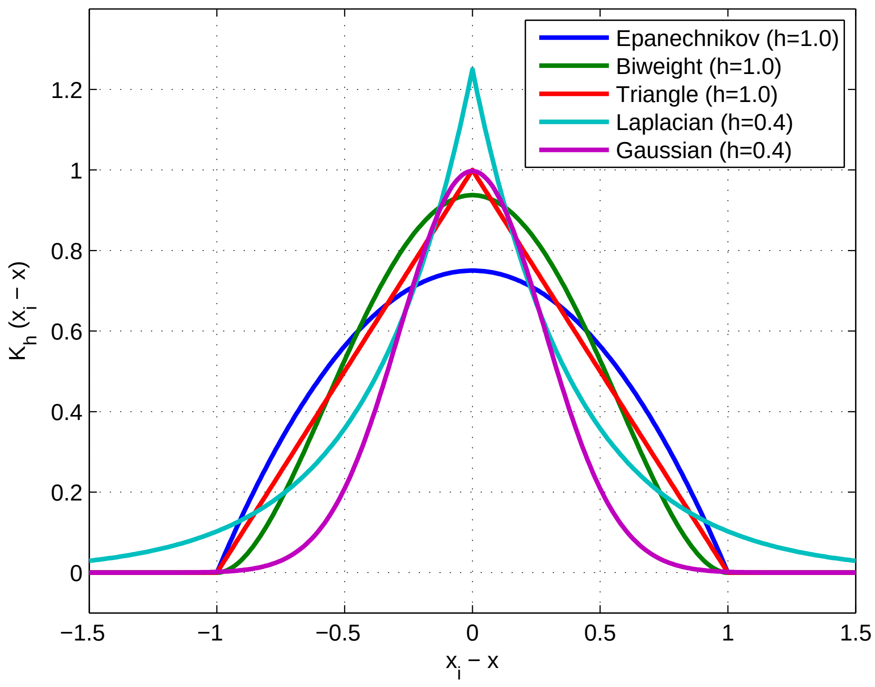
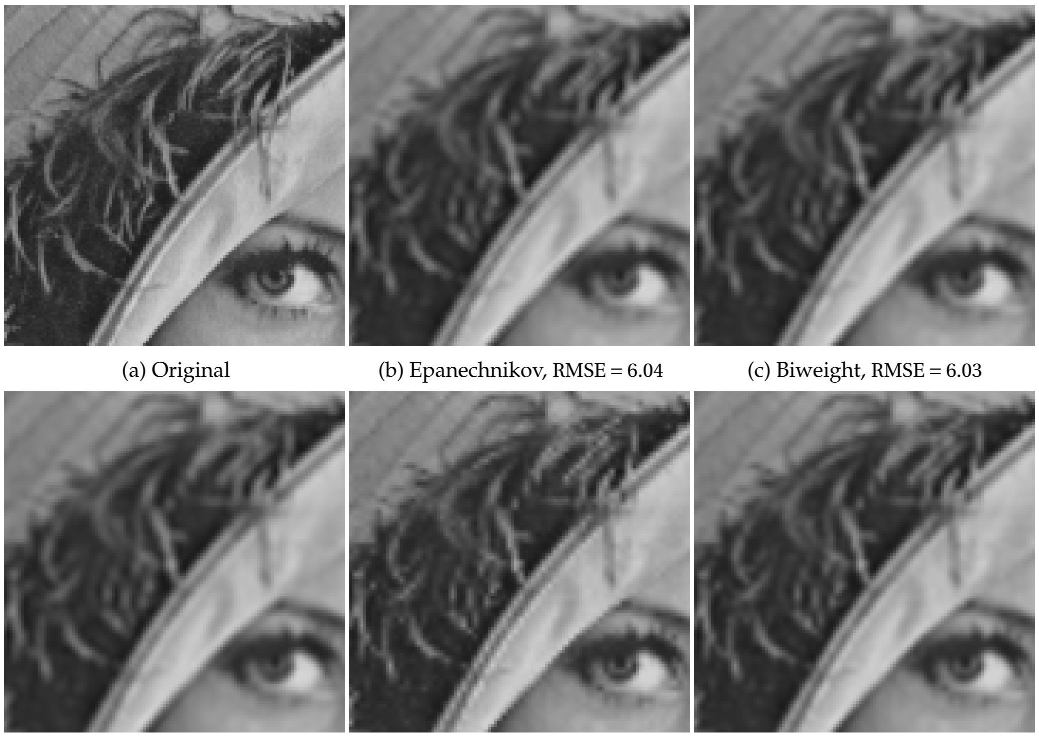


Figure 6.1: A comparison of the possible kernel functions.

part of the original images. Then, we upscaled the downsampled image by the second order classic kernel regression with different kernel functions. The upscaled results using Epanechnikov, biweight, triangle, Laplacian, and Gaussian functions are shown in Figures **6.2**(b)-(f) and **6.3**(b)-(f), respectively. In these image upscaling examples with classic kernel regression, although Laplacian kernel function numerically outperforms the other kernel functions, there are some staircase artifacts around edges. It would be interesting to examine these kernel functions in the data-adaptive kernel regression approach in terms of the numerical and visual performance.

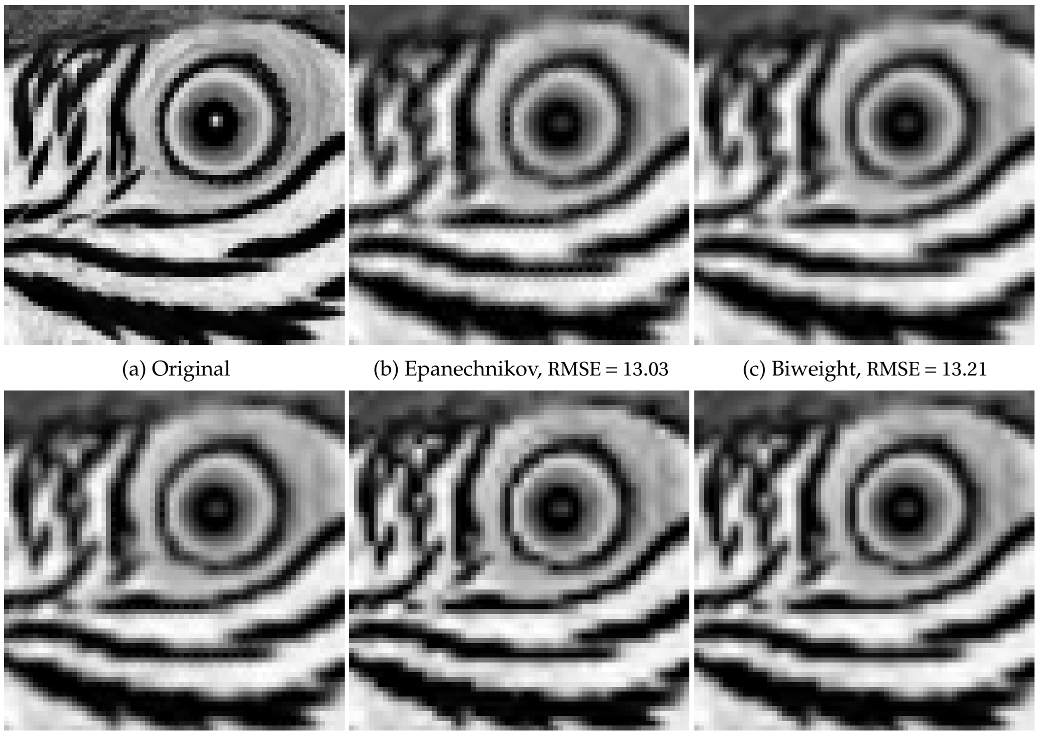

(a) Original (b) Epanechnikov, RMSE = 6.04 (c) Biweight, RMSE = 6.03

(d) Triangle, RMSE = 6.23 (e) Laplacian, RMSE = 5.22 (f) Gaussian, RMSE = 5.53

Figure 6.2: The performance comparison of a vaerity of kernel functions for image upscaling. In this example, we downsampled the original Lena image shown in (a) with the factor of 2:1, and then upscale the downsampled image with the factor of 1:2. The upscaled images by Epanechnikov, biweight, triangle, Laplacian, and Gaussian kernel functions are shown in (b)-(f), respectively. The smoothing parameters are optimized by the cross-validation method.

6.2.2 The Choice of Distance Metric

Having introduced the possible choices for kernel function in Table **6.1**, we see that all the kernel functions are defined in terms of the distance between two vectors (i.e. $\mathbf{x}_i - \mathbf{x}$). In this section, we discuss a possibility of improving the performance of kernel regression by the choice of the distance metric. A starting point can be found in pattern classification/recognition [115, 116], where the choice of the distance metric is an important issue, e.g., for the classification task. Table **6.2** shows some representative



(a) Original (b) Epanechnikov, RMSE = 13.03 (c) Biweight, RMSE = 13.21

(d) Triangle, RMSE = 12.57 (e) Laplacian, RMSE = 12.00 (f) Gaussian, RMSE = 12.11

Figure 6.3: The performance comparison of a vaerity of kernel functions by image upscaling. In this example, we downsampled the original Parrot image shown in (a) with the factor of 2:1, and then upscale the downsampled image with the factor of 1:2. The upscaled images by Epanechnikov, biweight, triangle, Laplacian, and Gaussian kernel functions are shown in (b)-(f), respectively. The smoothing parameters are optimized by the cross-validation method.

distance metrics in pattern recognition with related image restoration methods, where $d(\boldsymbol{v}_i, \boldsymbol{v}_j)$ represents the distance between two vectors $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$. The distance metrics are categorized into two types: non-adaptive and adaptive. Euclidean, Manhattan, and Minkowski (L_2, L_1, and L_p-norm, respectively) are categorized as non adaptive distance metrics. These metrics are typically most effective when their choice matches the statistical distribution of the data being treated. For instance, when the data are corrupted by white Gaussian noise, Euclidean distance is an appropriate choice. It should be noted that, in our steering kernel approach, we compute the weight values based on the distances between pixels measured by a data-adaptive metric.

Standardized Euclidean, Mahalanobis, and Bhattacharyya metrics are categorized as adaptive distance metrics. Standardized Euclidean distance can take local noise variance into account. Mahalanobis distance is one of the frequently used metrics in pattern recognition, and it measures the distance between samples with the scatter (covariance) matrix. The scatter matrix tells that how the given samples distribute and it is helpful in classifying when the scatter matrix of a cluster is similar to the other clusters'. If that is not the case, Bhattacharyya distance is more effective, since it takes the differences of the scatter matrices into account. Thus, when we compute the distance between two vectors which belong to different clusters, Bhattacharyya distance becomes large.

Bilateral kernel function (2.2) and steering kernel function (2.9) are closely related to standardized Euclidean distance and Mahalanobis distance, respectively. When we use Gaussian function for the both radiometric and spatial kernels, the bilateral kernel function can be expressed with standardized Euclidean distance as

$$\begin{aligned} K_{h_\mathrm{s}}(\mathbf{x}_i - \mathbf{x}_j) K_{h_\mathrm{r}}(y_i - y_j) &= \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}\|_2^2}{h_\mathrm{s}^2}\right) \exp\left(-\frac{\left|y_i - y_j\right|}{h_\mathrm{r}^2}\right) \\ &= \exp\left\{-\left(\boldsymbol{v}_i - \boldsymbol{v}_j\right)^T \mathbf{D}^{-1} \left(\boldsymbol{v}_i - \boldsymbol{v}_j\right)\right\} \\ &= \exp\left(-d_\mathrm{s}^2(\boldsymbol{v}_i, \boldsymbol{v}_j; \mathbf{D})\right), \end{aligned} \tag{6.1}$$

where $\boldsymbol{v}_i = [y_i, \mathbf{x}_i^T]^T$ and $\mathbf{D} = \mathrm{diag}\{h_\mathrm{r}^2, h_\mathrm{s}^2, h_\mathrm{s}^2\}$. For steering kernel regression, unlike the Mahalanobis distance with a fixed scatter matrix shown in Table **6.2**, we compute the scatter matrix $\boldsymbol{\Sigma}$ of Mahalanobis distance for each pair of pixels by the covariance matrix of local gradient vectors. For instance, using the same definition $\boldsymbol{v}_i = [y_i, \mathbf{x}_i^T]^T$, the

Table 6.2: Representative Distance Metrics and Related Image Restoration Methods

Metric	Definition	Note	Related methods
Euclidean	$d_{L_2}(\boldsymbol{v}_i,\boldsymbol{v}_j)=\left\|\boldsymbol{v}_i-\boldsymbol{v}_j\right\|_2$	L_2-norm	Classic KR [3], NLM [57], UINTA [117]
Manhattan	$d_{L_1}(\boldsymbol{v}_i,\boldsymbol{v}_j)=\left\|\boldsymbol{v}_i-\boldsymbol{v}_j\right\|_1$	L_1-norm	Total variation
Minkowski	$d_{Lp}(\boldsymbol{v}_i,\boldsymbol{v}_j)=\left\|\boldsymbol{v}_i-\boldsymbol{v}_j\right\|_p$	L_p-norm	
Standardized Euclidean	$d_{\mathrm{S}}^2(\boldsymbol{v}_i,\boldsymbol{v}_j)=(\boldsymbol{v}_i-\boldsymbol{v}_j)^T\mathbf{D}_i^{-1}(\boldsymbol{v}_i-\boldsymbol{v}_j)$	$\mathbf{D}_i\in\mathscr{R}^{M\times M}$ is a diagonal matrix given by $\mathrm{diag}\{\mathrm{var}(v_1),\cdots,\mathrm{var}(v_M)\}$ where $\{v_m\}_{m=1}^M$ are the elements of $\boldsymbol{v}_i$.	Bilateral filter [7, 8], OSA [118]
Mahalanobis	$d_{\mathrm{M}}^2(\boldsymbol{v}_i,\boldsymbol{v}_j)=(\boldsymbol{v}_i-\boldsymbol{v}_j)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{v}_i-\boldsymbol{v}_j)$	$\boldsymbol{\Sigma}\in\mathscr{R}^{M\times M}$ is the scatter (covariance) matrix given by $\frac{1}{P}\sum_\ell(\boldsymbol{v}_\ell-\overline{\boldsymbol{v}})(\boldsymbol{v}_\ell-\overline{\boldsymbol{v}})^T$ with ℓ for all the given sample where $\overline{\boldsymbol{v}}$ is the mean vector of $\{\boldsymbol{v}_\ell\}_{\ell=1}^P$.	Steering KR
Bhattacharyya	$d_{\mathrm{B}}^2(\boldsymbol{v}_i,\boldsymbol{v}_j)=\frac{1}{8}(\boldsymbol{v}_i-\boldsymbol{v}_j)^T\left(\frac{\boldsymbol{\Sigma}_i+\boldsymbol{\Sigma}_j}{2}\right)^{-1}(\boldsymbol{v}_i-\boldsymbol{v}_j)+\frac{1}{2}\ln\frac{\left\|\frac{\boldsymbol{\Sigma}_i+\boldsymbol{\Sigma}_j}{2}\right\|}{\sqrt{\|\boldsymbol{\Sigma}_i\|\|\boldsymbol{\Sigma}_j\|}}$	$\boldsymbol{\Sigma}_i$ and $\boldsymbol{\Sigma}_j$ are the within-cluster scatter matrices of the clusters that the vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_j$ belong respectively.	

2-D steering kernel function can be expressed as

$$\begin{aligned} K_{\mathbf{H}_i}(\mathbf{x}_i-\mathbf{x}_j) &= \frac{1}{2\pi\sqrt{\det\mathbf{H}_i}}\exp\left\{-\frac{1}{2}\left(\mathbf{x}_i-\mathbf{x}_j\right)^T\mathbf{H}_i^T\mathbf{H}_i\left(\mathbf{x}_i-\mathbf{x}_j\right)\right\} \\ &= \frac{1}{2\pi\sqrt{\det\mathbf{H}_i}}\exp\left\{-\frac{1}{2}d_{\mathrm{M}}^2(\boldsymbol{v}_i,\boldsymbol{v}_j;\boldsymbol{\Sigma}_i)\right\}, \end{aligned} \tag{6.2}$$

where

$$\boldsymbol{\Sigma}_i=\begin{bmatrix} 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{H}_i^T\mathbf{H}_i \end{bmatrix}. \tag{6.3}$$

Therefore, the proposed steering kernel function can be regarded as one of the many other possible combinations of the kernel functions in Table **6.1** and the distance metrics in Table **6.2**.

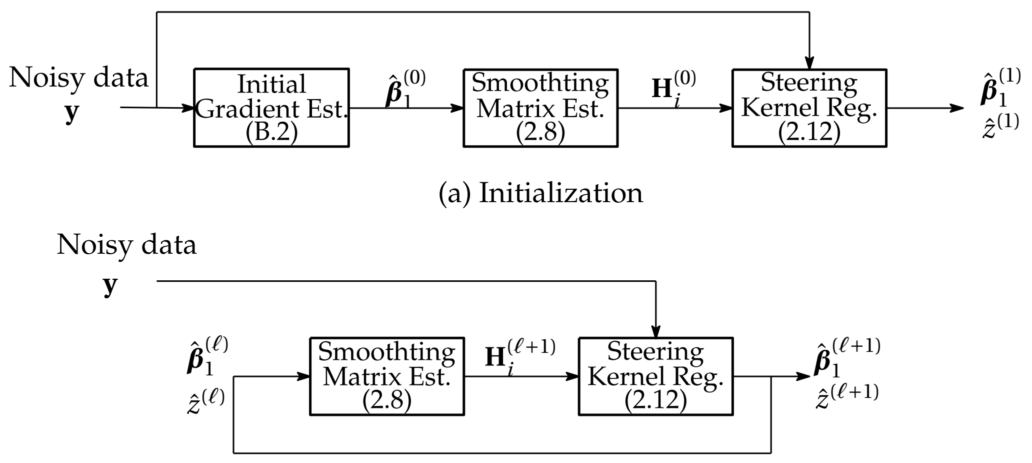

Noisy data
$\mathbf{y}$
Initial Gradient Est. (B.2)
$\hat{\boldsymbol{\beta}}_1^{(0)}$
Smoothting Matrix Est. (2.8)
$\mathbf{H}_i^{(0)}$
Steering Kernel Reg. (2.12)
$\hat{\boldsymbol{\beta}}_1^{(1)}$
$\hat{z}^{(1)}$
(a) Initialization
Noisy data
$\mathbf{y}$
$\hat{\boldsymbol{\beta}}_1^{(\ell)}$
$\hat{z}^{(\ell)}$
Smoothting Matrix Est. (2.8)
$\mathbf{H}_i^{(\ell+1)}$
Steering Kernel Reg. (2.12)
$\hat{\boldsymbol{\beta}}_1^{(\ell+1)}$
$\hat{z}^{(\ell+1)}$
(b) Iteration

Figure 6.4: Block diagram representation of iterative steering kernel regression: (a) the initialization process, and (b) the iteration process where we apply SKR to the given noisy data.

6.2.3 Iteration Filtering Scheme

We proposed an iterative filtering algorithm in Figure **2.12**, where we update the steering kernel weights and apply SKR to the previous estimate, in order to improve the smoothing performance. The iterative filtering algorithm is just one way to improve the estimate, and there are more possible iterative implementations (e.g. [119]). Figure **6.4** shows a possible iterative filtering algorithm where we update the steering kernel weights and apply SKR to the original noisy data in each iteration. Although the iterative filtering algorithms shown in Figures **2.12** and **6.4** are similar, their numerical behaviors are very different. Using a selected region of Lena image shown in Figure **2.15**(a), we analyze their behavior with respect to MSE, bias, and variance by Monte-Carlo simulations for the task of image denoising with three different smoothing parameters for each iterative filtering algorithm. In this experiment, we added white Gaussian noise with $\sigma = 25$ to the cropped Lena image. The MSE, vari-

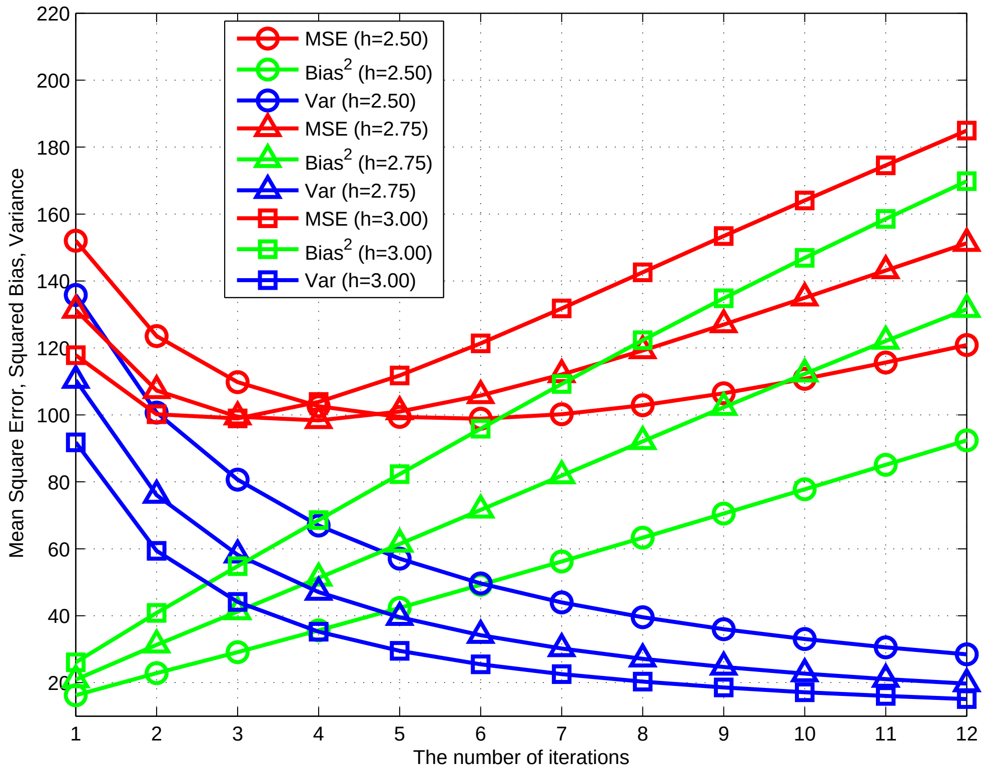


Figure 6.5: An example of the behavior of mean square error, variance, and bias of the iterative steering kernel regression proposed in Figure **2.12** with three different smoothing parameters $h = 2.5, 2.75$, and 3.0.

ance and squared bias of both iterative filtering algorithms are shown in Figures **6.5** and **6.6**. The MSE value of the iterative filtering algorithm of Figure **2.12** decreases in the first several iterations and then starts increasing. The iterative filtering asymptotically produces a flat image unless we stop the iteration. On the other hand, the MSE value of the alternative iterative filtering algorithm of Figure **6.4** also decreases in the first few iterations, but the MSE value converges to a constant value. There are two interesting properties in both iterative filtering algorithms: (i) for each case, even as we vary the smoothing parameter (h), the minimum MSE value is about the same, and (ii) the MSE value becomes the smallest when the variance and the squared bias are equal.

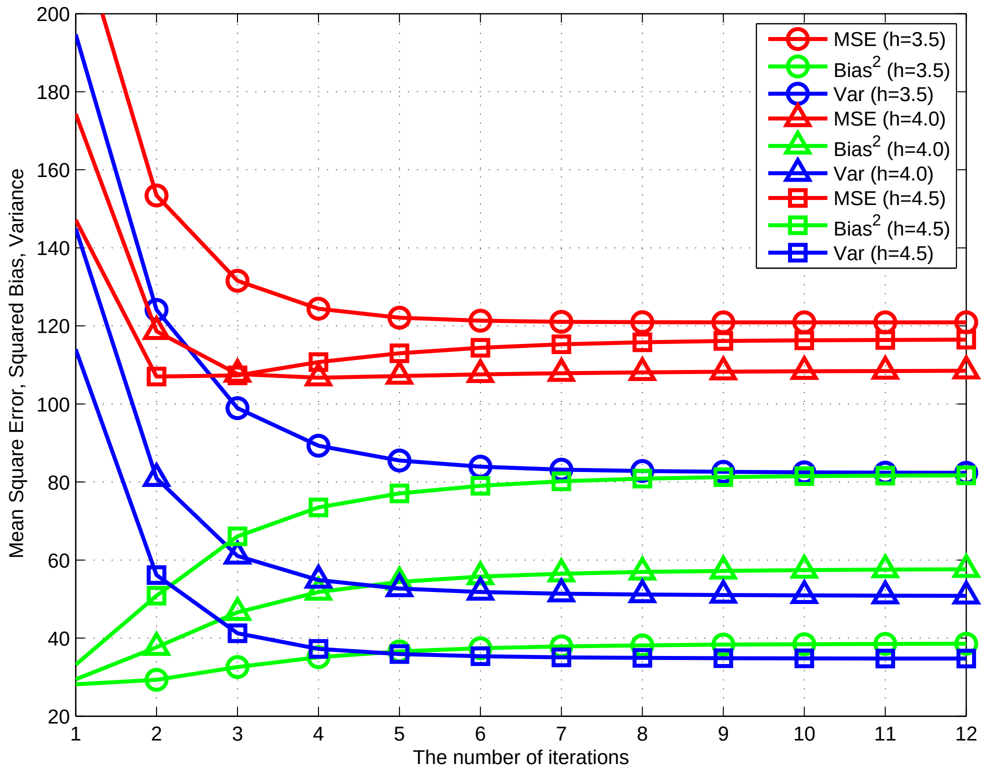


Figure 6.6: An example of the behavior of mean square error, variance, and bias of the iterative steering kernel regression shown in Figure **6.4** with three different smoothing parameters $h = 3.5, 4.0$, and 4.5.

Correspondingly, in the case where the variance and squared bias become asymptotically parallel (the curve for $h = 3.5$ in Figure **6.6**), then the corresponding MSE value approaches a minimum only after infinitely many iterations.

The questions that one needs to address in the iterative filtering approach are the following: (i) what would be the preferable behavior of the MSE, variance, and bias? (it worth noting that in the limited experiments we showed above, the iterative filtering scheme in Figure **2.12**, consistently produces a smaller MSE value than the one in Figure **6.4**), and (ii) how do we optimize the number of iterations to reach the lowest MSE? One possible way to optimize the number of iterations is to measure the quality

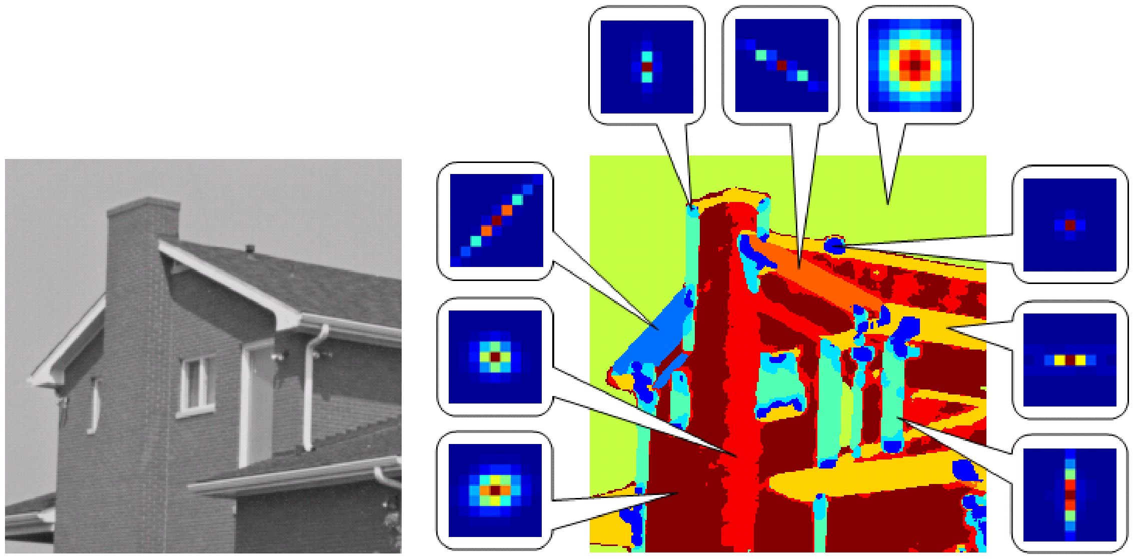

House image (256 × 256) (b) Centroids of the clustered LSKs and the corresponding pixels

Figure 6.7: A quantization example of local steering kernel (LSK): (a) the original House image, and (b) centroids of the clustered LSKs by K-means method and the corresponding pixels.

of the estimated image after each iteration by, for example, metric Q [95]. If the image quality becomes worse than the previous estimate, we stop the iteration.

6.2.4 Quantization of Local Steering Kernel

In order to reduce the computational load, in Section 4.2.4.3, we generate a compact look-up table of the steering matrices and select a steering matrix from the table for each sample. Although the singular value decomposition of the local gradient vectors is avoided, we still need to compute the local steering kernel weights W_i in (2.12). A more desirable way would be to select a set of local steering kernel (LSK) weights directly from a look-up table using the steering kernel parameters (i.e. the scaling, elongation, and angle parameters) or the naive covariance matrix of the local gradient vectors (2.10) as the parameters.

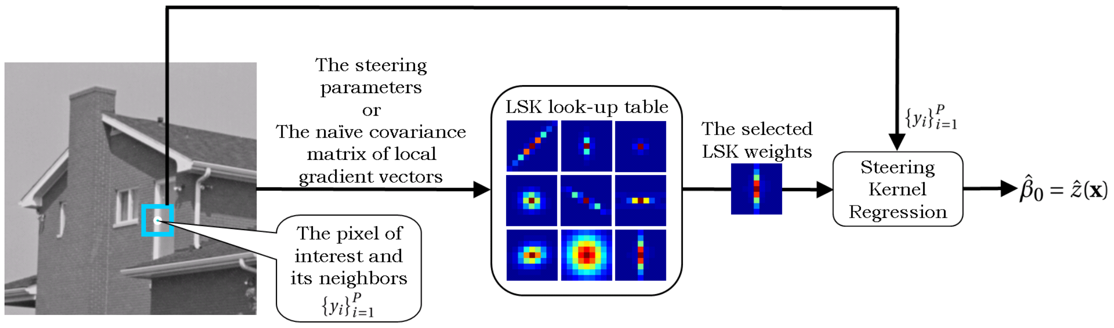

The steering parameters or The naive covariance matrix of local gradient vectors
The pixel of interest and its neighbors $\{y_i\}_{i=1}^P$
LSK look-up table
The selected LSK weights
$\{y_i\}_{i=1}^P$
Steering Kernel Regression
$\hat{\beta}_0 = \hat{z}(\mathbf{x})$

Figure 6.8: A diagram of the steering kernel regression with a look-up table of local steering kernel weights.

A desired look-up table is compact, but it produces high quality output images. Of course we could create a general look-up table of LSKs for any images, but it would be better to take the content of the underlying image into account in terms of the visual quality of the output images. Figure **6.7** shows a possible way to learn an efficient set of LSKs where we first compute the LSKs for all the pixels of the House image and then classify the LSKs. In this example, we classified the LSKs into 9 clusters by the k-means algorithm. The centroid of each cluster is a learned LSK. Figure **6.7**(b) illustrates the learned LSKs and the pixels with the corresponding LSKs. For example, when we treat the flat area (i.e. the sky of the house image), the LSK equally spreading to every direction is an appropriate choice.

Now, the 9 LSKs are the entries of the look-up table. Using the look-up table, Figure **6.8** shows a diagram of the SKR with learned LSKs. For example, when smoothing a pixel located on a vertical structure, first we compute its steering parameters or the naive covariance matrix of the local gradient vectors. Using them as the parameters, we select a suitable LSK from the look-up table. Once we have LSK weights, the weighted average of the neighboring pixels $\{y_i\}$ with the LSK is our pixel estimate $\hat{z}(\mathbf{x})$.

Advantages of the LSK quantization by classification are (i) the quantization is helpful in generating a stable set of LSKs; when the given image carries strong noise or is missing many pixels, it is difficult to have good estimates of orientations, and (ii) we could avoid updating the LSKs in the iterative filtering. Once the stable set of LSKs is available, we can use them again. The questions that we need to address in this LSK quantization are (i) how to optimize the parameter to compute LSK for each pixel and (ii) how to classify the LSKs so that we have an efficient and low complexity representation of the set.

6.2.5 Kernel Regression for Vector Functions

When processing color images, we applied SKR to the three color channels separately, while using YCbCr coordinates in order to reduce color artifacts. However, since one color pixel is a vector quantity, i.e. it consists of three values:

$$\mathbf{z}(\mathbf{x}_i) = \begin{bmatrix} z_{\mathrm{R}}(\mathbf{x}_i) \\ z_{\mathrm{G}}(\mathbf{x}_i) \\ z_{\mathrm{B}}(\mathbf{x}_i) \end{bmatrix}, \quad \text{or} \quad \begin{bmatrix} z_{\mathrm{Y}}(\mathbf{x}_i) \\ z_{\mathrm{Cb}}(\mathbf{x}_i) \\ z_{\mathrm{Cr}}(\mathbf{x}_i) \end{bmatrix}, \tag{6.4}$$

the data model should be

$$\mathbf{y}_i = \mathbf{z}(\mathbf{x}_i) + \boldsymbol{\varepsilon}_i \tag{6.5}$$

for color images. Using the local approximation between the color pixel of interest and its neighbors by Taylor series, we have the kernel regression for the vector function

$$\hat{\mathbf{z}}(\mathbf{x}) = \arg\min_{\mathbf{z}(\mathbf{x})} \sum_i \left[\mathbf{y}_i - \mathbf{z}(\mathbf{x})\right]^2 K_{\mathbf{H}_i}(\mathbf{x}_i - \mathbf{x}), \tag{6.6}$$

where

$$\mathbf{z}(\mathbf{x}) = \mathbf{z}(\mathbf{x}_i) - \left.\frac{\partial^T \mathbf{z}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_i} (\mathbf{x}_i - \mathbf{x}) - \cdots. \tag{6.7}$$

Now, since the color pixels are vector quantities, we could compute the dissimilarity between vectors $\mathbf{y}_i$ and $\mathbf{z}(\mathbf{x})$ by the cross product (i.e. $\mathbf{y}_i \times \mathbf{z}(\mathbf{x})$). In [120, 47], Keren *et al.* and Farsiu *et al.* proposed an intercolor dependency which smooths the pixel of interest with its neighboring pixels minimizing the cross products of the pixel of interest and its neighbors. We can regard their approach as the zeroth order kernel regression with the cross product distance measure, and the kernel regression for vector function (6.6) is a generalization and probably a more appropriate way to process color images.

Appendix A

Equivalent Kernels

Study of (1.33) shows that $\mathbf{X}^T\mathbf{K}\mathbf{X}$ is a $(N+1)\times(N+1)$ block matrix, with the following structure:

$$\mathbf{X}^T\mathbf{K}\mathbf{X} = \begin{bmatrix} a_{11} & \mathbf{a}_{12} & \mathbf{a}_{13} & \cdots \\ \mathbf{a}_{21} & \mathbf{a}_{22} & \mathbf{a}_{23} & \cdots \\ \mathbf{a}_{31} & \mathbf{a}_{32} & \mathbf{a}_{33} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}, \tag{A.1}$$

where $\mathbf{a}_{lm}$ is an $l \times m$ matrix for the 2-D case. The block elements of (A.1) for orders up to $N = 2$ are as follows:

$$a_{11} = \sum_{i=1}^{P} K_{\mathbf{H}}(\mathbf{x}_1 - \mathbf{x}), \tag{A.2}$$

$$\mathbf{a}_{12} = \mathbf{a}_{21}^T = \sum_{i=1}^{P} (\mathbf{x}_1 - \mathbf{x})^T K_{\mathbf{H}}(\mathbf{x}_1 - \mathbf{x}), \tag{A.3}$$

$$\mathbf{a}_{22} = \sum_{i=1}^{P} (\mathbf{x}_1 - \mathbf{x})(\mathbf{x}_1 - \mathbf{x})^T K_{\mathbf{H}}(\mathbf{x}_1 - \mathbf{x}), \tag{A.4}$$

$$\mathbf{a}_{13} = \mathbf{a}_{31}^T = \sum_{i=1}^{P} \text{vech}^T\left\{(\mathbf{x}_1 - \mathbf{x})(\mathbf{x}_1 - \mathbf{x})^T\right\} K_{\mathbf{H}}(\mathbf{x}_1 - \mathbf{x}), \tag{A.5}$$

$$\mathbf{a}_{23} = \mathbf{a}_{32}^T = \sum_{i=1}^{P} (\mathbf{x}_1 - \mathbf{x})\text{vech}^T\{(\mathbf{x}_1 - \mathbf{x})(\mathbf{x}_1 - \mathbf{x})^T\} K_{\mathbf{H}}(\mathbf{x}_1 - \mathbf{x}), \tag{A.6}$$

$$\mathbf{a}_{33} = \sum_{i=1}^{P} \text{vech}\{(\mathbf{x}_1 - \mathbf{x})(\mathbf{x}_1 - \mathbf{x})^T\}\text{vech}^T\{(\mathbf{x}_1 - \mathbf{x})(\mathbf{x}_1 - \mathbf{x})^T\} K_{\mathbf{H}}(\mathbf{x}_1 - \mathbf{x}). \tag{A.7}$$

With the above shorthand notations, the equivalent kernel functions $W_i(\cdot)$ in (1.35) for up to $N = 2$ are given by

$$W_i(K, \mathbf{H}, N{=}0, \mathbf{x}_i - \mathbf{x}) = \frac{K_{\mathbf{H}}(\mathbf{x}_i - \mathbf{x})}{a_{11}}, \tag{A.8}$$

$$W_i(K, \mathbf{H}, N{=}1, \mathbf{x}_i - \mathbf{x}) = \frac{\left\{1 - \mathbf{a}_{12}\mathbf{a}_{22}^{-1}(\mathbf{x}_i - \mathbf{x})\right\} K_{\mathbf{H}}(\mathbf{x}_i - \mathbf{x})}{a_{11} - \mathbf{a}_{12}\mathbf{a}_{22}^{-1}\mathbf{a}_{21}}, \tag{A.9}$$

$$W_i(K, \mathbf{H}, N{=}2, \mathbf{x}_i - \mathbf{x}) = \frac{\left[1 - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}(\mathbf{x}_i - \mathbf{x}) - \mathbf{A}_{13}\mathbf{A}_{33}^{-1}\text{vech}\{(\mathbf{x}_1 - \mathbf{x})(\mathbf{x}_1 - \mathbf{x})^T\}\right] K_{\mathbf{H}}(\mathbf{x}_i - \mathbf{x})}{a_{11} - \mathbf{A}_{12}\mathbf{A}_{22}^{-1}\mathbf{a}_{21} - \mathbf{A}_{13}\mathbf{A}_{33}^{-1}\mathbf{a}_{31}} \tag{A.10}$$

with

$$\mathbf{A}_{12} = \mathbf{a}_{12} - \mathbf{a}_{13}\mathbf{a}_{33}^{-1}\mathbf{a}_{32}, \quad \mathbf{A}_{22} = \mathbf{a}_{22} - \mathbf{a}_{23}\mathbf{a}_{33}^{-1}\mathbf{a}_{32},$$

$$\mathbf{A}_{13} = \mathbf{a}_{13} - \mathbf{a}_{12}\mathbf{a}_{22}^{-1}\mathbf{a}_{23}, \quad \mathbf{A}_{33} = \mathbf{a}_{33} - \mathbf{a}_{32}\mathbf{a}_{22}^{-1}\mathbf{a}_{23}. \tag{A.11}$$

Appendix B

Local Gradient Estimation

In this appendix, we formulate the estimation of the direct gradient $\boldsymbol{\beta}_1$ of the second order kernel regressor ($N = 2$). Note that direct gradient estimation is useful not only for the iterative steering kernel regression, presented in Section 2.1.3, but also for many diverse applications such as estimating motion via gradient-based methods (e.g., optical flow) and dominant orientation estimation.

In the solution of the optimization of kernel regression (1.32), the second and third element of $\widehat{\mathbf{b}}$ give the estimate of local gradient:

$$\nabla \hat{z}(\mathbf{x}) = \hat{\boldsymbol{\beta}}_1 = \begin{bmatrix} \mathbf{e}_2^T \\ \mathbf{e}_3^T \end{bmatrix} \left(\mathbf{X}^T \mathbf{K} \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{K}\, \mathbf{y}, \tag{B.1}$$

where $\mathbf{e}_2$ and $\mathbf{e}_3$ are column vectors with the second and third elements equal to one, and the rest equal to zero. Using the notations of (A.3)-(A.7) in Appendix A, the local quadratic gradient estimator is expressed as

$$\nabla \hat{z}(\mathbf{x}) = \sum_{i=1}^{P} \mathbb{B}^{-1} \left[-\mathbf{B}_{21} B_{11}^{-1} + (\mathbf{x}_i - \mathbf{x}) - \mathbf{B}_{23} \mathbf{B}_{33}^{-1} \text{vech}\left\{ (\mathbf{x}_i - \mathbf{x})(\mathbf{x}_i - \mathbf{x})^T \right\} \right] K_{\mathbf{H}}(\mathbf{x}_i - \mathbf{x})\, y_i, \tag{B.2}$$

where

$$B_{11} = a_{11} - \mathbf{a}_{13}\mathbf{a}_{33}^{-1}\mathbf{a}_{31}, \quad \mathbf{B}_{21} = \mathbf{a}_{21} - \mathbf{a}_{22}\mathbf{a}_{33}^{-1}\mathbf{a}_{31},$$

$$\mathbf{B}_{23} = \mathbf{a}_{23} - \mathbf{a}_{21}a_{11}^{-1}\mathbf{a}_{13}, \quad \mathbf{B}_{33} = \mathbf{a}_{33} - \mathbf{a}_{31}a_{11}^{-1}\mathbf{a}_{13},$$

$$\mathbb{B} = \mathbf{a}_{22} - \mathbf{B}_{21}B_{11}^{-1}\mathbf{a}_{12} - \mathbf{B}_{23}\mathbf{B}_{33}^{-1}\mathbf{a}_{32}. \tag{B.3}$$

Appendix C

Kernel-Based Deblurring Estimator

Differentiating the cost function of the kernel-based deblurring method (3.19), we have

$$\frac{\partial C(\mathbb{U}_N)}{\partial \mathbb{U}_N} = \frac{\partial C_{\mathrm{L}}(\mathbb{U}_N)}{\partial \mathbb{U}_N} + \lambda \frac{\partial C_{\mathrm{R}}(\mathbb{U}_N)}{\partial \mathbb{U}_N}. \tag{C.1}$$

Using (3.14) and (3.18), the terms in the right hand side are

$$\begin{aligned}\frac{\partial C_{\mathrm{R}}(\mathbb{U}_N)}{\partial \mathbb{U}_N} &= \sum_{\nu_1=-\zeta}^{\zeta} \sum_{\nu_2=-\zeta}^{\zeta} \frac{\partial}{\partial \mathbb{U}_N} \left\| \{\mathbf{W}_u(\boldsymbol{\nu})\}^{\frac{1}{q}} \left(\mathbb{E}_0 \mathbb{U}_N - \mathbf{S}_{x_1}^{\nu_1} \mathbf{S}_{x_2}^{\nu_2} \mathbb{I}_N \mathbb{U}_N\right) \right\|_q^q \\ &= \sum_{\nu_1=-\zeta}^{\zeta} \sum_{\nu_2=-\zeta}^{\zeta} \left(\mathbb{E}_0 - \mathbf{S}_{x_1}^{\nu_1} \mathbf{S}_{x_2}^{\nu_2} \mathbb{I}_N\right)^T \mathbf{W}_u(\boldsymbol{\nu}) \left\{ \mathrm{sign}\left(\underline{\mathbf{u}} - \mathbf{S}_{x_1}^{\nu_1} \mathbf{S}_{x_2}^{\nu_2} \mathbb{I}_N \mathbb{U}_N\right) \odot \left|\underline{\mathbf{u}} - \mathbf{S}_{x_1}^{\nu_1} \mathbf{S}_{x_2}^{\nu_2} \mathbb{I}_N \mathbb{U}_N\right|^{q-1} \right\}\end{aligned} \tag{C.2}$$

$$\begin{aligned}\frac{\partial C_{\mathrm{L}}(\mathbb{U}_N)}{\partial \mathbb{U}_N} &= \sum_{\nu_1=-\zeta}^{\zeta} \sum_{\nu_2=-\zeta}^{\zeta} \frac{\partial}{\partial \mathbb{U}_N} \left\| \{\mathbf{W}_z(\boldsymbol{\nu})\}^{\frac{1}{2}} \left(\underline{\mathbf{y}} - \mathbf{S}_{x_1}^{\nu_1} \mathbf{S}_{x_2}^{\nu_2} \mathbb{G}_N \mathbb{U}_N\right) \right\|_2^2 \\ &= \sum_{\nu_1=-\zeta}^{\zeta} \sum_{\nu_2=-\zeta}^{\zeta} \mathbb{G}_N^T \mathbf{S}_{x_1}^{-\nu_1} \mathbf{S}_{x_2}^{-\nu_2} \mathbf{W}_z(\boldsymbol{\nu}) \left(\underline{\mathbf{y}} - \mathbf{S}_{x_1}^{\nu_1} \mathbf{S}_{x_2}^{\nu_2} \mathbb{G}_N \mathbb{U}_N\right),\end{aligned} \tag{C.3}$$

where $\odot$ is the element-by-element multiplication operator for two vectors, and $\mathbb{E}_0$ is a block matrix with identity matrix for the first block and zero matrices for the rest, i.e. $\mathbb{E}_0 = [\mathbf{I}, \mathbf{0}, \cdots, \mathbf{0}]$ and $\underline{\mathbf{u}} = \mathbb{E}_0 \mathbb{U}_N$. We also note that the transpose of the shift operator is equivalent to the shift back operator, i.e. $\{\mathbf{S}_{x_1}^{\nu_1}\}^T = \mathbf{S}_{x_1}^{-\nu_1}$.

Appendix D

Motion Estimation

For motion estimation between two images (an anchor frame and the target frame), taking into account that the forward (from the anchor frame to the target frame) and the backward motion (from the target frame to the anchor frame) should be equal in magnitude and opposite in direction, we define the brightness consistency equation (BCE) [91] as

$$\mathbf{J}\,\mathbf{m} + \mathbf{J}_t = \underline{\mathbf{0}}, \tag{D.1}$$

where $\mathbf{m} = [m_1, m_2]$ is the translational motion vector of interest, and

$$\mathbf{J} = \begin{bmatrix} \underline{\mathbf{z}}_{x_1}^{(n)} & \underline{\mathbf{z}}_{x_2}^{(n)} \\ \underline{\mathbf{z}}_{x_1}^{(n')} & \underline{\mathbf{z}}_{x_2}^{(n')} \end{bmatrix}, \quad \mathbf{J}_t = \begin{bmatrix} \underline{\mathbf{z}}_t \\ \underline{\mathbf{z}}_t \end{bmatrix} \tag{D.2}$$

with $\underline{\mathbf{z}}_{x_1}^{(n)}$ is the gradients of the n-th frame in column-stack vector form and $\underline{\mathbf{z}}_t$ is the temporal gradient (difference); $\underline{\mathbf{z}}_t = \underline{\mathbf{z}}^{(n)} - \underline{\mathbf{z}}^{(n')}$. We use least square estimator to find the motion parameter m_1 and m_2:

$$\widehat{\mathbf{m}} = \arg\min_{\mathbf{m}} \|\mathbf{J}\,\mathbf{m} + \mathbf{J}_t\|_2^2, \tag{D.3}$$

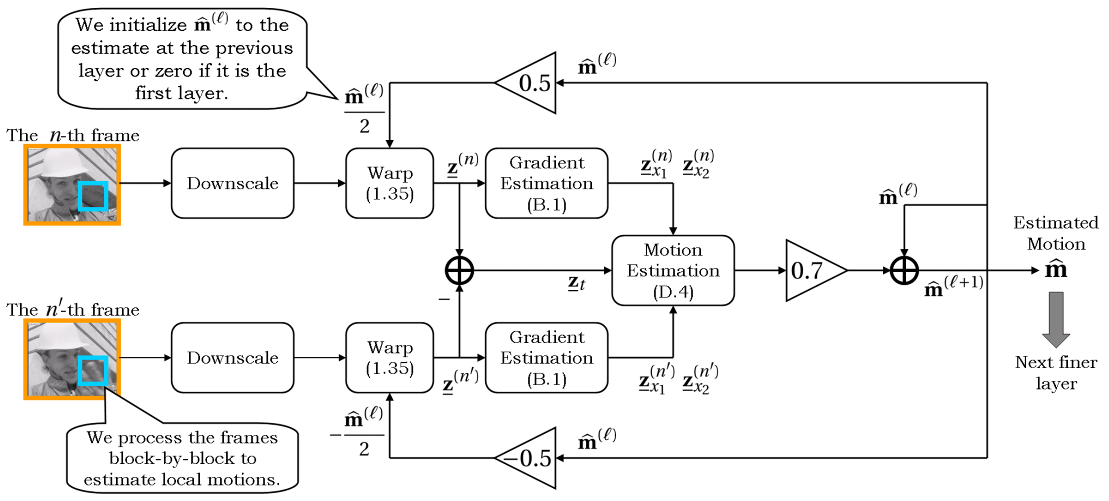

We initialize $\widehat{\mathbf{m}}^{(\ell)}$ to the estimate at the previous layer or zero if it is the first layer.
The n-th frame
The n'-th frame
We process the frames block-by-block to estimate local motions.
Downscale
Downscale
Warp (1.35)
Warp (1.35)
$\frac{\widehat{\mathbf{m}}^{(\ell)}}{2}$
$-\frac{\widehat{\mathbf{m}}^{(\ell)}}{2}$
0.5
−0.5
$\widehat{\mathbf{m}}^{(\ell)}$
$\underline{\mathbf{z}}^{(n)}$
$\underline{\mathbf{z}}^{(n')}$
Gradient Estimation (B.1)
Gradient Estimation (B.1)
$\underline{\mathbf{z}}_{x1}^{(n)}$ $\underline{\mathbf{z}}_{x2}^{(n)}$
$\underline{\mathbf{z}}_{x1}^{(n')}$ $\underline{\mathbf{z}}_{x2}^{(n')}$
$\underline{\mathbf{z}}_t$
−
Motion Estimation (D.4)
0.7
$\widehat{\mathbf{m}}^{(\ell+1)}$
Estimated Motion
$\widehat{\mathbf{m}}$
Next finer layer

Figure D.1: The block diagram representation of our motion estimation.

which yields the motion estimator

$$\widehat{\mathbf{m}} = -\left(\mathbf{J}^T\mathbf{J}\right)^{-1}\mathbf{J}^T\mathbf{J}_t. \tag{D.4}$$

Since the BCE (D.1) becomes a more accurate expression when the amplitude of the motion vector is smaller, we estimate accurate motions iteratively with the multiscale technique as shown in Figure **D.1**. First, we downscale the anchor and target frames ($\underline{\mathbf{z}}^{(n)}$ and $\underline{\mathbf{z}}^{(n')}$) to build a multiscale pyramid, and then start motion estimation from the coarse resolution layer because the motion becomes smaller in the coarse layer. Next, in order to make the displacement between the frames even smaller, we warp the anchor and the target frames by $0.5\widehat{\mathbf{m}}^{(\ell)}$ and $-0.5\widehat{\mathbf{m}}^{(\ell)}$, respectively, where $\widehat{\mathbf{m}}^{(\ell)}$ is the previous motion estimate ($\widehat{\mathbf{m}}^{(0)} = \mathbf{0}$), using the second order classic kernel regression (1.35). We warp both the anchor and target frames with the same amount of subpixel displacements so that the influence of the blur effects, caused by the interpolator (1.35), can be reduced for the motion estimation. After warping, we compute the spatial gradients of the warped frames by (B.1) and estimate the leftover motion components by (D.4). We scale the estimated leftover motion by 0.7 (a damping factor) to stabilize the iteration process, and then update the previous motion estimate. The iteration process continues until the estimated motion converges[1]. Then we begin the iteration process in a finer resolution layer initializing the motion vector $\widehat{\mathbf{m}}^{(0)}$ to the estimated motion vector at the previous layer. When estimating local motions, we apply the iterative motion estimation to the frames block-by-block.

[1]In case the iteration process may not converge, we set the maximum number of iterations in order not to repeat the process forever.

Appendix E

3-D Steering Kernel Parameters

Using the (compact) SVD (4.30) of the local gradient vector $\mathbf{J}_i$ (4.27), we can express the naive estimate of steering matrix as:

$$\begin{aligned}
\widehat{\mathbf{C}}_i^{\text{naive}} &= \mathbf{J}_i^T \mathbf{J}_i = \mathbf{V}_i \mathbf{S}_i^T \mathbf{S}_i \mathbf{V}_i^T \\
&= \mathbf{V}_i \operatorname{diag}\left\{s_1^2, s_2^2, s_3^2\right\} \mathbf{V}_i^T \\
&= s_1 s_2 s_3 \mathbf{V}_i \operatorname{diag}\left\{\frac{s_1}{s_2 s_3}, \frac{s_2}{s_1 s_3}, \frac{s_3}{s_1 s_2}\right\} \mathbf{V}_i^T \\
&= Q\gamma_i \left[\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\right] \operatorname{diag}\left\{\varrho_1, \varrho_2, \varrho_3\right\} \left[\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\right]^T \\
&= Q\gamma_i \sum_{q=1}^{3} \varrho_q \mathbf{v}_q \mathbf{v}_q^T, \qquad \text{(E.1)}
\end{aligned}$$

where

$$\varrho_1 = \frac{s_1}{s_2 s_3}, \quad \varrho_2 = \frac{s_2}{s_1 s_3}, \quad \varrho_3 = \frac{s_3}{s_1 s_2}, \quad \gamma_i = \frac{s_1 s_2 s_3}{Q}, \qquad \text{(E.2)}$$

and Q is the number of rows in $\mathbf{J}_i$. Since the singular values (s_1, s_2, s_3) may become zero, we regularized the elongation parameters (ϱ_q) and the scaling parameter (γ_i) as shown in (4.29) from being zero.

Bibliography

[1] T. F. Chan and J. Shen, "Nontexture inpainting by curvature-driven diffusions," *Journal of Visual Communication and Image Representation*, vol. 12, no. 10, pp. 436–449, May 2001.

[2] S. Farsiu, D. Robinson, M. Elad, and P. Milanfar, "Fast and robust multi-frame super-resolution," *IEEE Transactions on Image Processing*, vol. 13, no. 10, pp. 1327–1344, October 2004.

[3] M. P. Wand and M. C. Jones, *Kernel Smoothing*, ser. Monographs on Statistics and Applied Probability. London; New York: Chapman and Hall, 1995.

[4] P. Yee and S. Haykin, "Pattern classification as an ill-posed, inverse problem: a reglarization approach," *Proceeding of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP*, vol. 1, pp. 597–600, April 1993.

[5] H. Knutsson and C. F. Westin, "Normalized and differential convolution - methods for interpolation and filtering of incomplete and uncertain data," *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Regocnition (CVPR)*, pp. 515–523, June 1993.

[6] T. Q. Pham, L. J. van Vliet, and K. Schutte, "Robust fusion of irregularly sampled data using adaptive normalized convolution," *EURASIP Journal on Applied Signal Processing, Article ID 83268*, 2006.

[7] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," *Proceeding of the 1998 IEEE International Conference of Compute Vision, Bombay, India*, pp. 836–846, January 1998.

[8] M. Elad, "On the origin of the bilateral filter and ways to improve it," *IEEE Transactions on Image Processing*, vol. 11, no. 10, pp. 1141–1150, October 2002.

[9] X. Li and M. T. Orchard, "New edge-directed interpolation," *IEEE Transactions on Image Processing*, vol. 10, no. 10, pp. 1521–1527, October 2001.

[10] N. K. Bose and N. Ahuja, "Superresolution and noise filtering using moving least squares," *IEEE Transactions on Image Processing*, vol. 15, no. 8, pp. 2239–2248, August 2006.

[11] B. W. Silverman, *Density Estimation for Statistics and Data Analysis*, ser. Monographs on Statistics and Applied Probability. London; New York: Chapman and Hall, 1986.

[12] W. Hardle, *Applied Nonparametric Regression*. Cambridge [England] ; New York: Cambride University Press, 1990.

[13] W. Hardle, M. Muller, S. Sperlich, and A. Werwatz, *Nonparametric and Semiparametric Models*, ser. Springer Series in Statistics. Berlin ; New York: Springer, 2004.

[14] W. Hardle, *Smooting Technique : with Implementation in S*, ser. Springer Series in Statistics. New York: Springer-Verlag, 1991.

[15] E. A. Nadaraya, "On estimating regression," *Theory of Probability and its Applications*, pp. 141–142, September 1964.

[16] D. Ruppert and M. P. Wand, "Multivariate locally weighted least squares regression," *The Annals of Statistics*, vol. 22, no. 3, pp. 1346–1370, September 1994.

[17] M. G. Schimek, *Smoothing and Regression -Approaches, Computation, and Application-*, ser. Wiley Series in Probability and Statistics. New York: Wiley-Interscience, 2000.

[18] M. Unser, "Splines: A perfect fit for signal and image processing," *IEEE Signal Processing Magazine*, vol. 16, no. 6, pp. 22–38, November 1999.

[19] J. E. Gentle, W. Hadle, and Y. Mori, *Handbook of Computational Statistics: Concepts and Methods*. Berlin ; New York: Springer, 2004, pp. 539–564 (Smoothing: Local Regression Techniques).

[20] C. de Boor, *A Practical Guide to Splines*. New York: Springer-Verlag, 1978.

[21] R. L. Eubank, *Nonparametric Regression and Spline Smoothing*, 2nd ed., ser. Statistics, Textbooks and Monographs. New York: Marcel Dekker, 1999.

[22] L. Piegl and W. Tiller, *The NURBS Book*. New York: Springer, 1995.

[23] M. Arigovindan, M. Suhling, P. Hunziker, and M. Unser, "Variational image reconstruction from arbitrarily spaced samples: A fast multiresolution spline solution," *IEEE Transactions on Image Processing*, vol. 14, no. 4, pp. 450–460, April 2005.

[24] N. S. Jayant and P. Noll, *Digital coding of waveforms : principles and applications to speech and video*, ser. Pretice Hall signal processing. New Jersey: Englewood Cliffs: Prentice Hall, 1984.

[25] M. J. Black, G. Sapiro, D. H. Marimont, and D. Heeger, "Robust anisotropic diffusion," *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 421–432, March 1998.

[26] Y. You and M. Kaveh, "Fourth-order partial differential equations for noise removal," *IEEE Transactions on Image Processing*, vol. 9, no. 10, pp. 1723–1730, October 2000.

[27] L. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D*, vol. 60, pp. 259–268, November 1992.

[28] S. M. Kay, *Fundamentals of Statistical Signal Processing - Estimation Theory -*, ser. Signal Processing Series. Englewood Cliffs, N.J.: PTR Prentice-Hall, 1993.

[29] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Upper Saddle River, N.J.: Prentice Hall, 2002.

[30] J. Fan and I. Gijbels, *Local Polynomial Modelling and its Applications*, ser. Monographs on Statistics and Applied Probability. London ; New york: Chapman and Hall, 1996.

[31] C. K. Chu and J. S. Marron, "Choosing a kernel regression estimator (with discussion)," *Statistical Science*, vol. 6, pp. 404–436, 1991.

[32] ——, "Comparison of two bandwidth selectors with dependent errors," *The Annals of Statistics*, vol. 4, pp. 1906–1918, 1991.

[33] W. Hardle and P. Vieu, "Kernel regression smoothing of time series," *Journal of Time Series Analysis*, vol. 13, pp. 209–232, 1992.

[34] V. Katkovnic, K. Egiazarian, and J. Astola, *Local Approximation Techniques in Signal and Image Processing*. Washington: SPIE – The International Society for Optical Engineering, 2006.

[35] K. Fukunaga and L. D. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition," *IEEE Transactions of Information Theory*, vol. 21, pp. 32–40, 1975.

[36] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, May 2002.

[37] S. M. Smith and J. M. Brady, "Susan – a new approach to low level image processing," *International Journal of Computer Vision*, vol. 23, no. 1, pp. 45–78, 1997.

[38] X. Feng and P. Milanfar, "Multiscale principal components analysis for image local orientation estimation," *Proceedings of the 36th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA*, November 2002.

[39] R. Mester and M. Muhlich, "Improving motion and orientation estimation using an equilibrated total least squares approach," *Proceedings of IEEE International Conference in Image Processing*, pp. 929–932, 2001.

[40] R. Mester, M. Hotter, and R. B. GmbH, "Robust displacement vector estimation including a statistical error analysis," *Proceedings of Fifth International Conference on Image Processing and its Applications*, pp. 168–172, 1995.

[41] K. V. Mardia, J. T. Kent, and J. M. Bibby, *Multivariate Analysis*. London ; New York: Academic Press, 1979.

[42] A. Edelman, "Eigenvalues and condition numbers of random matrices," *SIAM Journal on Matrix Analysis and Aplications*, vol. 9, pp. 543–560, 1988.

[43] H. Seo and P. Milanfar, "Training-free, generic object detection using locally adaptive regression kernels," *To appear in IEEE Transactions on Pattern Analysis and Machine Intteligence*, 2010.

[44] M. Aharon, M. Elad, and A. Bruckstein, "The k-svd: An algorithm for designing of overcomplete dictionaries for sparse representation," *IEEE Transactions on Signal Processing*, vol. 54, no. 11, pp. 4311–4322, November 2006.

[45] K. Dabov, A. Foi, V. Katkovnic, and K. Egiazarian, "Image denoising by sparse 3D transform-domain collaborative filtering," *IEEE Transactions of Image Processing*, vol. 16, no. 8, pp. 2080–2095, August 2007.

[46] J. Portilla, V. Strela, M. Wainwright, and E. P. Simoncelli, "Image denoising using scale mixtures of Gaussians in the wavelet domain," *IEEE Transactions on Image Processing*, vol. 12, no. 11, pp. 1338–1351, November 2003.

[47] S. Farsiu, M. Elad, and P. Milanfar, "Multiframe demosacing and super-resolution of color images," *IEEE Transactions on Image Processing*, vol. 15, no. 1, pp. 141–159, January 2006.

[48] F. R. Hampel, E. M. Ronchetti, and P. J. Rousseeuw, *Robust Statistics: The Approach Based on Influence Functions*. New York: Wiley, 1986.

[49] P. J. Huber, *Robust Statistics*. New York: Wiley, 1981.

[50] M. J. Black and P. Anandan, "The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields," *Computer Vision and Image Understanding*, vol. 63, no. 1, pp. 75–104, January 1996.

[51] H. Knutsson and C.-F. Westin, "Normalized and differential convolution: Methods for interpolation and filtering of incomplete and uncertain data," *Proceedings of Computer Vision and Pattern Recognition (Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition'93)*, pp. 515–523, June 16-19 1993.

[52] H. Takeda, S. Farsiu, and P. Milanfar, "Kernel regression for image processing and reconstruction," *IEEE Transactions on Image Processing*, vol. 16, no. 2, pp. 349–366, February 2007.

[53] V. Katkovnic, K. Egiazarian, and J. Astola, "A spatially adaptive nonparametric regression image deblurring," *IEEE Transactions on Image Processing*, vol. 14, no. 10, pp. 1469–1478, October 2005.

[54] A. N. Tikhonov, A. Goncharsky, V. V. Stepanov, and A. G. Yagola, *Numerical Meth-*

ods for the Solution of Ill-Posed Problems. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1995.

[55] S. Osher and L. I. Rudin, "Feature-oriented image enhancement using shock filters," *SIAM Journal on Numerical Analysis*, vol. 27, no. 4, pp. 919–940, August 1990.

[56] T. F. Chan, S. Osher, and J. Shen, "The digital TV filter and nonlinear denoising," *IEEE Transactions of Image Processing*, vol. 10, no. 2, pp. 231–241, February 2001.

[57] A. Buades, B. Coll, and J. M. Morel, "A review of image denosing algorithms, with a new one," *Multiscale Modeling and Simulation, Society for Industrial and Applied Mathematics (SIAM) Interdisciplinary Journal*, vol. 4, no. 2, pp. 490–530, 2005.

[58] H. Takeda, S. Farsiu, and P. Milanfar, "Higher order bilateral filter ant its properties," *Proceedings of the SPIE Conference on Computational Imaging, San Jose, CA*, January 2007.

[59] R. Neelamani, H. Choi, and R. Baraniuk, "ForWaRD: Fourier-wavelet regularized deconvolution for ill-conditioned systems," *IEEE Transactions on Signal Processing*, vol. 52, no. 2, pp. 418–433, February 2004.

[60] P. Thibault, M. Dierolf, A. Menzel, O. Bunk, C. David, and F. Pfeiffer, "High-resolution scanning x-ray diffraction microscopy," *Science*, vol. 321, pp. 379–382, July 2008, issue 5887.

[61] X. Li and M. T. Orchard, "New edge-directed interpolation," *IEEE Transactions of Image Processing*, vol. 10, no. 10, pp. 1521–1527, October 2001.

[62] D. P. Mitchell and A. N. Netravali, "Reconstruction filters in computer graphics," *Computer Graphics*, vol. 22, no. 4, pp. 221–228, August 1988.

[63] W. T. Freeman, T. R. Jones, and E. C. Pasztor, "Example-based super resolution," *IEEE Computer Graphics*, vol. 22, no. 2, pp. 56–65, March/April 2002.

[64] S. Kanumuri, O. G. Culeryuz, and M. R. Civanlar, “Fast super-resolution reconstructions of mobile video using warped transforms and adaptive thresholding,” *Proceedings of SPIE*, vol. 6696, p. 66960T, 2007.

[65] M. Elad and Y. Hel-Or, “A fast super-resolution reconstruction algorithm for pure translational motion and common space-invariant blur,” *IEEE Transactions on Image Processing*, vol. 10, no. 8, pp. 1187–1193, August 2001.

[66] H. Fu and J. Barlow, “A regularized structured total least squares algorithm for high-resolution image reconstruction,” *Linear Algebra and its Applications*, vol. 391, pp. 75–98, November 2004.

[67] M. Irani and S. Peleg, “Super resolution from image sequence,” *Proceedings of 10th International Conference on Pattern Recognition (ICPR)*, vol. 2, pp. 115–120, 1990.

[68] M. K. Ng, J. Koo, and N. K. Bose, “Constrained total least squares computations for high resolution image reconstruction with multisensors,” *International Journal of Imaging Systems and Technology*, vol. 12, pp. 35–42, 2002.

[69] P. Vandewalle, L. Sbaiz, M. Vetterli, and S. Susstrunk, “Super-resolution from highly undersampled images,” *Proceedings of International Conference on Image Processing (ICIP)*, pp. 889–892, September 2005, Genova, Italy.

[70] N. A. Woods, N. P. Galatsanos, and A. K. Katsaggelos, “Stochastic methods for joint registration, restoration, and interpolation of multiple undersampled images,” *IEEE Transactions on Image Processing*, vol. 15, no. 1, pp. 201–213, January 2006.

[71] A. Zomet, A. Rav-Acha, and S. Peleg, “Robust super-resolution,” *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001, Hawaii.

[72] R. Hardie, “A fast image super-resolution algorithm using an adaptive Wiener filter,” *IEEE Transactions on Image Processing*, vol. 16, no. 12, pp. 2953–2964, December 2007.

[73] S. Park, M. Park, and M. Kang, “Super-resolution image reconstruction: A technical overview,” *IEEE Signal Processing Magazine*, vol. 20, no. 3, pp. 21–36, May 2003.

[74] S. Fujiwara and A. Taguchi, “Motion-compensated frame rate up-conversion based on block matching algorithm with multi-size blocks,” *Proceedings of International Symposium on Intelligent Signal Processing and Communication Systems*, pp. 353–356, December 2005.

[75] A. Huang and T. Q. Nguyen, “A multistage motion vector processing method for motion-compensated frame interpolation,” *IEEE Transactions on Image Processing*, vol. 17, no. 5, pp. 694–708, May 2008.

[76] S. Kang, K. Cho, and Y. Kim, “Motion compensated frame rate up-conversion using extended bilateral motion estimation,” *IEEE Transactions on Consumer Electronics*, vol. 53, pp. 1759–1767, November 2007.

[77] B. Choi, J. Han, C. Kim, and S. Ko, “Motion-compensated frame interpolation using bilateral motion estimation and adaptive overlapped block motion compensation,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 4, pp. 407–416, April 2007.

[78] J. van de Weijer and R. van den Boomgaard, “Least squares and robust estimation of local image structure,” *Scale Space. International Conference*, vol. 2695, no. 4, pp. 237–254, 2003.

[79] K. S. Ni, S. Kumar, N. Vasconcelos, and T. Q. Nguyen, “Single image super-resolution based on support vector regression,” *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 2, May 2006.

[80] M. Protter, M. Elad, H. Takeda, and P. Milanfar, “Generalizing the non-local-means to super-resolution reconstruction,” *IEEE Transactions on Image Processing*, vol. 16, no. 2, pp. 36–51, January 2009.

[81] M. Protter and M. Elad, “Super-resolution with probabilistic motion estimation,” *IEEE Transactions on Image Processing*, vol. 18, no. 8, pp. 1899–1904, August 2009.

[82] A. Danielyan, A. Foi, V. Katkovnik, and K. Egiazarian, “Image and video super-resolution via spatially adaptive block-matching filtering,” *Proceedings of International Workshop on Local and Non-Local Approximation in Image Processing (LNLA)*, August 2008, Lausanne, Switzerland.

[83] A. Gersho and R. M. G. and, *Vector Quantization and Signal Compression*. Boston: Kluwer Academic Publishers, 1992.

[84] B. Lucas and T. Kanade, “An iterative image registration technique with an application to sterio vision,” *Proceedings of DARPA Image Understanding Workshop*, pp. 121–130, 1981.

[85] C. Stiller and J. Konrad, “Estimating motion in image sequences - a tutorial on modeling and computation of 2d motion,” *IEEE Signal Processing Magagine*, vol. 16, pp. 70–91, July 1999.

[86] M. Ozkan, M. I. Sezan, and A. M. Tekalp, “Adaptive motion-compensated filtering of noisy image sequences,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 3, no. 4, pp. 277–290, August 2003.

[87] R. M. Haralick, “Edge and region analysis for digital image data,” *Computer Graphic and Image Processing (CGIP)*, vol. 12, no. 1, pp. 60–73, January 1980.

[88] K. Q. Weinberger and G. Tesauro, “Metric learning for kernel regression,” *Proceedings of the Eleventh International Workshop on Artificial Intelligence and Statistics*, pp. 608–615, 2007, (AISTATS-07), Puerto Rico.

[89] J. J. Gibson, *The Perception of the Visual World*. Boston: Houghton Mifflin, 1950.

[90] D. N. Lee and H. Kalmus, "The optic flow field: the foundation of vision," *Philosophical Transactions of the Royal Society of London Series B-Biological Sciences*, vol. 290, no. 1038, pp. 169–179, 1980.

[91] B. K. Horn, *Robot Vision*. Cambridge: MIT Press, 1986.

[92] J. Wright and R. Pless, "Analysis of persistent motion patterns using the 3d structure tensor," *Proceedings of the IEEE Workshop on Motion and Video Computing*, 2005.

[93] S. Chaudhuri and S. Chatterjee, "Performance analysis of total least squares methods in three-dimensional motion estimation," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 5, pp. 707–714, October 1991.

[94] H. Takeda, H. Seo, and P. Milanfar, "Statistical approaches to quality assessment for image restoration," *Proceedings of the International Conference on Consumer Electronics*, January 2008, Las Vegas, NV, Invited paper.

[95] X. Zhu and P. Milanfar, "Automatic parameter selection for denoising algorithms using a no-reference measure of image content," submitted to IEEE Transactions on Image Processing.

[96] G. Farnebäck, "Polynomial expansion for orientation and motion estimation," Ph.D. dissertation, Linköping University, Sweden, SE-581 83 Linköping, Sweden, 2002, dissertation No 790, ISBN 91-7373-475-6.

[97] S. Zhu and K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Transactions on Image Processing*, vol. 9, no. 2, pp. 287–290, February 2000.

[98] R. Fergus, B. Singh, A. Hertzmann, S. T. Roweis, and W. Freeman, "Removing

camera shake from a single photograph," *ACM Transactions on Graphics*, vol. 25, pp. 787–794, 2006.

[99] Q. Shan, J. Jia, and A. Agarwala, "High-quality motion deblurring from a single image," *ACM Transactions on Graphics*, vol. 27, pp. 73:1–73:10, 2008.

[100] M. Ben-Ezra and S. K. Nayar, "Motion-based motion deblurring," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 689–698, June 2004.

[101] Y. Tai, H. Du, M. S. Brown, and S. Lin, "Image/video deblurring using a hybrid camera," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2008, Anchorage, AK.

[102] S. Cho, Y. Matsushita, and S. Lee, "Removing non-uniform motion blur from images," *Proceedings of IEEE 11th International Conference on Computer Vision (ICCV)*, October 2007, Rio de Janeiro, Brazil.

[103] A. Levin, "Blind motion deblurring using image statistics," *The Neural Information Processing Systems (NIPS)*, 2006.

[104] P. Milanfar, "Projection-based, frequency-domain estimation of superimposed translational motions," *Journal of the Optical Society of America: A, Optics and Image Science*, vol. 13, no. 11, pp. 2151–2162, November 1996.

[105] ——, "Two dimensional matched filtering for motion estimation," *IEEE Transactions on Image Processing*, vol. 8, no. 3, pp. 438–444, March 1999.

[106] D. Robinson and P. Milanfar, "Fast local and global projection-based methods for affine motion estimation," *Journal of Mathematical Imaging and Vision (Invited paper)*, vol. 18, pp. 35–54, January 2003.

[107] ——, "Fundamental performance limits in image registration," *IEEE Transactions on Image Processing*, vol. 13, no. 9, pp. 1185–1199, September 2004.

[108] H. Ji and C. Liu, "Motion blur identification from image gradients," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2008, Anchorage, AK.

[109] S. Dai and Y. Wu, "Motion from blur," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2008, Anchorage, AK.

[110] J. Chen, L. Yuan, C. Tang, and L. Quan, "Robust dual motion deblurring," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2008, Anchorage, AK.

[111] E. Borissoff, "Optimal temporal sampling aperture for HDTV varispeed acquisition," *SMPTE Motion Imageging Journal*, vol. 113, no. 4, pp. 104–109, 2004.

[112] E. Shechtman, Y. Caspi, and M. Irani, "Space-time super-resolution," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 4, pp. 531–545, April 2005.

[113] B. Wilburn, N. Joshi, V. Vaish, M. Levoy, and M. Horowitz, "High-speed videography using a dense camera array," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 294–301, 2004, Washington, DC.

[114] A. Huang and T. Nguyen, "Correlation-based motion vector processing with adaptive interpolation scheme for motion-compensated frame interpolation," *IEEE Transactions on Image Processing*, vol. 18, no. 4, pp. 740–752, April 2009.

[115] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2nd edition)*. New York, NY: Wiley Interscience, 2001.

[116] S. Theodoridis and K. Koutroumbas, *Pattern Recognition (4th edition)*. San Diego, CA: Academic Press, 2009.

[117] S. P. Awate and R. T. Whitaker, "Unsupervised, information-theoretic, adaptive

image filtering for image restoration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 3, pp. 364–376, March 2006.

[118] C. Kervrann and J. Boulanger, "Optimal spatial adapation for patch-based image denoising," *IEEE Transactions on Image Processing*, vol. 15, no. 10, October 2006.

[119] M. Charest and P. Milanfar, "On iterative regularization and its application," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 3, pp. 406–411, March 2008.

[120] D. Keren and M. Osadchy, "Restoring subsamled color images," *Machine Vision and Applications*, vol. 11, no. 4, pp. 197–202, December 1999.