3

Locally Adaptive Kernel Regression for Space-Time Super-Resolution

Hiroyuki Takeda

University of California, Santa Cruz, USA

Peyman Milanfar

University of California, Santa Cruz, USA

CONTENTS

Abstract — In this chapter, we discuss a novel framework for adaptive enhancement and spatio-temporal upscaling of videos containing complex motions. Our approach is based on multidimensional kernel regression, where each pixel in the video sequence is approximated with a 3-D local (Taylor) series, capturing the essential local behavior of its spatiotemporal neighborhood. The coefficients of this series are estimated by solving a local weighted least-squares problem, where the weights are a function of the 3-D space-time orientation in the neighborhood. As this framework is fundamentally based upon the comparison of neighboring pixels in both space and time, it implicitly contains information about the local motion of the pixels across time, therefore rendering unnecessary an explicit computation of motions of modest size. When large motions are present, a basic, rough motion compensation

step returns the sequence to a form suitable again for motion-estimation-free super-resolution. The proposed approach not only significantly widens the applicability of super-resolution methods to a broad variety of video sequences containing complex motions, but also yields improved overall performance. Using several examples, we illustrate that the developed algorithm has super-resolution capabilities that provide improved optical resolution in the output, while being able to work on general input video with essentially arbitrary motion.

3.1 Introduction

The emergence of high definition displays in recent years (e.g. 720×1280 and 1080×1920 or higher spatial resolution, and up 240`Hz` 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 existing literature on enhancement and upscaling (sometimes called super-resolution[1]) is vast and rapidly growing in both the single frame case [9, 17] and the multi-frame (video) case [6, 8, 10, 12, 16, 24, 36, 39, 43], and many new algorithms for this problem have been proposed recently. Yet, one of the most fundamental roadblocks has 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 [6, 8, 10, 12, 16, 24, 36, 39, 43, 27]. As a result, most methods fail to perform well in the presence of complex motions which are quite common. 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.

In this work, we address the challenging problem of spatiotemporal video super-resolution in a fundamentally different way, which removes the need for explicit subpixel accuracy motion estimation. 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 [35], [31]. Classical kernel regression is a well-studied, non-parametric point estimation procedure. In our earlier work [31], we generalized the use of these techniques to spatially

[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".

adaptive (steering) kernel regression, which produces results that preserve and restore details with minimal assumptions on local signal and noise models [37]. Other related non-parametric techniques for multidimensional signal processing have emerged in recent years as well. In particular, the concept of normalized convolution [19], and the introduction of support vector machines [26] are notable examples. In the present work, the steering techniques in [31] are extended to 3-D where, as we will demonstrate, we can perform high fidelity space-time upscaling and super-resolution. Most importantly, this is accomplished without the explicit need for accurate motion estimation.

In a related work [28], we have generalized the non-local means (NLM) framework [2] 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, as explained in the Chapter by Protter and Elad. 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 recent example of a related approach appears in [5] where Danielyan, et al. have presented an extension of the block-matching 3-D filter (BM3D) [4] 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, our NLM-based approach [28], and Video-BM3D [5] 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. More specifically, existing methods have required highly accurate subpixel motion estimation and have thus failed to achieve resolution enhancement on arbitrary sequences.

We propose a framework which encompasses both video denoising, spatiotemporal upscaling, and super-resolution in 3-D. This 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. To be more specific, the computation of the filter coefficients is carried out in the following distinct steps. First, the local (spatiotemporal) gradients in the window of interest are used to calculate a covariance matrix, sometimes referred to as the "local structure tensor" [18]. This covariance matrix, which captures a locally dominant orientation *at each pixel*, is then used to define a local metric for measuring the similarity between the pixels in the neighborhood. This local metric distance is then inserted into a (Gaussian) kernel which, with proper normalization, then defines the local weights to be applied in the neighborhood.

The above approach is based on the concept of *steering kernel regression* (SKR), earlier introduced in [31] for images. A specific extension of these concepts to 3-D signals for the express purpose of video denoising and resolution enhancement are the main subjects of this chpater. As we shall see, since the development in 3-D involves the computation of orientation in space-time [13], motion information is implicitly and reliably captured. Therefore, unlike conventional approaches to video processing, 3-D SKR does not require explicit estimation of (modestly sized but essentially arbitrarily complex) motions, as this information is implicitly captured within 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. [38].

Naturally, the performance of the proposed approach is closely correlated with the quality of estimated space-time orientations. In the presence of noise, aliasing, and other artifacts, the estimates of orientation may not be initially accurate enough, and as we explain in Section 3.2.3, we therefore propose an iterative mechanism for estimating the orientations, which relies on the estimate of the pixels from the previous iteration.

To be more specific, as shown in Figure 3.8, 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 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 3.2.4, some level of explicit motion estimation is unavoidable to reduce temporal aliasing and achieve resolution enhancement. However, as we will illustrate in this chapter, this motion estimation can be quite rough (accurate to within a whole pixel at best). 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, our 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.

This chapter is organized as follows: in Section 3.2, first we briefly describes the fundamental concepts behind the SKR framework in 2-D and present the extension of the SKR framework to 3-D including discussions of how our method captures local complex motions and performs rough motion compensation, and explicitly describe its iterative implementation. In Section 3.3, we provide some experimental examples with both synthetic and real video sequences, and we conclude this chapter in Section 3.4.

3.2 Adaptive Kernel Regression

In this section, we first review the fundamental framework of *kernel regression* (KR) [37] and its extension, the *steering* kernel regression (SKR) [31], in 2-D. Then, we extend the steering approach to 3-D and discuss some important aspects of the 3-D extension.

3.2.1 Classic Kernel Regression in 2-D

The KR framework defines its data model as

$$y_i = z(\boldsymbol{x}_i) + \varepsilon_i, \quad \boldsymbol{x}_i \in \omega, \quad i = 1, \cdots, P, \tag{3.1}$$

where y_i is a noise-ridden sample measured at $\boldsymbol{x}_i = [x_{1i}, x_{2i}]^T$ (Note: x_{1i} and x_{2i} are spatial coordinates), $z(\,\cdot\,)$ is the (hitherto unspecified) *regression function* of interest, ε_i is an *i.i.d.* zero mean noise, and P is the total number of samples in an arbitrary "window" ω around a position $\boldsymbol{x}$ of interest as illustrated in Figure. 3.1. As such, the KR framework provides a rich mechanism for computing point-wise estimates of the regression function with minimal assumptions about global signal or noise models.

While the particular form of $z(\,\cdot\,)$ may remain unspecified, we can develop a generic local expansion of the function about a sampling point $\boldsymbol{x}_i$. Specifically, if the position of interest $\boldsymbol{x}$ is near the sample at $\boldsymbol{x}_i$, we have the N-th order

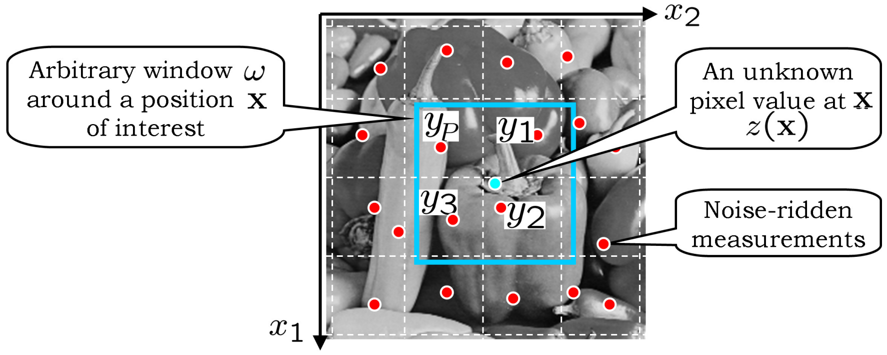

x_2
x_1
Arbitrary window ω around a position $\mathbf{x}$ of interest
An unknown pixel value at $\mathbf{x}$ $z(\mathbf{x})$
Noise-ridden measurements
y_P
y_1
y_3
y_2

FIGURE 3.1: The data model for the kernel regression framework

Taylor series

$$\begin{aligned} z(\boldsymbol{x}_i) &\approx z(\boldsymbol{x}) + \{\boldsymbol{\nabla} z(\boldsymbol{x})\}^T(\boldsymbol{x}_i - \boldsymbol{x}) + \frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{x})^T\{\boldsymbol{\mathcal{H}} z(\boldsymbol{x})\}^T(\boldsymbol{x}_i - \boldsymbol{x}) + \cdots \\ &\approx \beta_0 + \boldsymbol{\beta}_1^T(\boldsymbol{x}_i - \boldsymbol{x}) + \boldsymbol{\beta}_2^T \mathrm{vech}\big\{(\boldsymbol{x}_i - \boldsymbol{x})(\boldsymbol{x}_i - \boldsymbol{x})^T\big\} + \cdots \end{aligned} \tag{3.2}$$

where $\boldsymbol{\nabla}$ and $\boldsymbol{\mathcal{H}}$ are the gradient (2×1) and Hessian (2×2) operators, respectively, and vech$(\,\cdot\,)$ is the half-vectorization operator that lexicographically orders the lower triangular portion of a symmetric matrix into a column-stacked vector. Furthermore, β_0 is $z(\boldsymbol{x})$, which is the signal (or pixel) value of interest, and the vectors $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$ are

$$\begin{aligned} \boldsymbol{\beta}_1 &= \left[\ \frac{\partial z(\boldsymbol{x})}{\partial x_1}, \quad \frac{\partial z(\boldsymbol{x})}{\partial x_2}\ \right]^T, \\ \boldsymbol{\beta}_2 &= \frac{1}{2}\left[\ \frac{\partial^2 z(\boldsymbol{x})}{\partial x_1^2}, \quad \frac{\partial^2 z(\boldsymbol{x})}{\partial x_1 \partial x_2}, \quad \frac{\partial^2 z(\boldsymbol{x})}{\partial x_2^2},\ \right]^T. \end{aligned} \tag{3.3}$$

Since this approach is based on *local* signal representations (i.e. Taylor series), a logical step to take is to estimate the parameters $\{\boldsymbol{\beta}_n\}_{n=0}^N$ using all the neighboring samples $\{y_i\}_{i=1}^P$ while giving the nearby samples higher weights than samples farther away. A weighted least-square formulation of the fitting problem capturing this idea is

$$\min_{\{\boldsymbol{\beta}_n\}_{n=0}^N} \sum_{i=1}^{P} \left[y_i - \beta_0 - \boldsymbol{\beta}_1^T(\boldsymbol{x}_i - \boldsymbol{x}) - \boldsymbol{\beta}_2^T \mathrm{vech}\big\{(\boldsymbol{x}_i - \boldsymbol{x})(\boldsymbol{x}_i - \boldsymbol{x})^T\big\} - \cdots\right]^2 \cdot K_{\boldsymbol{H}}(\boldsymbol{x}_i - \boldsymbol{x}) \tag{3.4}$$

with

$$K_{\boldsymbol{H}}(\boldsymbol{x}_i - \boldsymbol{x}) = \frac{1}{\det(\boldsymbol{H})} K(\boldsymbol{H}^{-1}(\boldsymbol{x}_i - \boldsymbol{x})), \tag{3.5}$$

where N is the regression order, $K(\,\cdot\,)$ is the kernel function (a radially symmetric function such as a Gaussian), and $\boldsymbol{H}$ is the smoothing (2×2) matrix which dictates the "footprint" of the kernel function. The simplest choice of the smoothing matrix is $\boldsymbol{H} = h\boldsymbol{I}$, where h is called the *global smoothing parameter*. The contour of the kernel footprint is perhaps the most important factor in determining the quality of estimated signals. For example, it is desirable to use kernels with large footprints in the smooth local regions to reduce the noise effects, while relatively smaller footprints are suitable in the edge and textured regions to preserve the underlying signal discontinuity. Furthermore, it is desirable to have kernels that adapt themselves to the local structure of the measured signal, providing, for instance, strong filtering along an edge rather than across it. This last point is indeed the motivation behind the *steering* KR framework [31] which we will review Section 3.2.2.

Returning to the optimization problem (3.4), regardless of the regression order (N), and the dimensionality of the regression function, we can rewrite it as the weighted least squares problem:

$$\widehat{\boldsymbol{b}} = \arg\min_{\boldsymbol{b}} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{b})^T \boldsymbol{K} (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{b}), \tag{3.6}$$

where

$$\boldsymbol{y} = \begin{bmatrix} y_1, & y_2, & \cdots, & y_P \end{bmatrix}^T, \quad \boldsymbol{b} = \begin{bmatrix} \beta_0, & \boldsymbol{\beta}_1^T, & \cdots, & \boldsymbol{\beta}_N^T \end{bmatrix}^T, \tag{3.7}$$

$$\boldsymbol{K} = \operatorname{diag}\begin{bmatrix} K_{\boldsymbol{H}}(\boldsymbol{x}_1 - \boldsymbol{x}), & K_{\boldsymbol{H}}(\boldsymbol{x}_2 - \boldsymbol{x}), & \cdots, & K_{\boldsymbol{H}}(\boldsymbol{x}_P - \boldsymbol{x}) \end{bmatrix} \tag{3.8}$$

and

$$\boldsymbol{X} = \begin{bmatrix} 1, & (\boldsymbol{x}_1 - \boldsymbol{x}), & \operatorname{vech}\left\{(\boldsymbol{x}_1 - \boldsymbol{x})(\boldsymbol{x}_1 - \boldsymbol{x})^T\right\}, & \cdots \\ 1, & (\boldsymbol{x}_2 - \boldsymbol{x}), & \operatorname{vech}\left\{(\boldsymbol{x}_2 - \boldsymbol{x})(\boldsymbol{x}_2 - \boldsymbol{x})^T\right\}, & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ 1, & (\boldsymbol{x}_P - \boldsymbol{x}), & \operatorname{vech}\left\{(\boldsymbol{x}_P - \boldsymbol{x})(\boldsymbol{x}_P - \boldsymbol{x})^T\right\}, & \cdots \end{bmatrix} \tag{3.9}$$

with "diag" defining a diagonal matrix. Using the notation above, the optimization (3.4) provides the weighted least square estimator:

$$\widehat{\boldsymbol{b}} = \left(\boldsymbol{X}^T \boldsymbol{K} \boldsymbol{X}\right)^{-1} \boldsymbol{X}^T \boldsymbol{K}\, \boldsymbol{y} \tag{3.10}$$

and the estimate of the signal (i.e. pixel) value of interest β_0 is given by a weighted *linear* combination of the nearby samples:

$$\hat{z}(\boldsymbol{x}) = \widehat{\beta}_0 = \boldsymbol{e}_1^T \widehat{\boldsymbol{b}} = \sum_{i=1}^{P} W_i(K, \boldsymbol{H}, N, \boldsymbol{x}_i - \boldsymbol{x})\, y_i \tag{3.11}$$

where $\boldsymbol{e}_1$ is a column vector with the first element equal to one and the rest equal to zero, $\sum_i W_i = 1$, and we call W_i the *equivalent kernel* weight function for y_i (q.v. [31] or [37] for more detail). For example, for zeroth order regression (i.e. $N = 0$), the estimator (3.11) becomes

$$\hat{z}(\boldsymbol{x}) = \widehat{\beta}_0 = \frac{\displaystyle\sum_{i=1}^{P} K_{\boldsymbol{H}}(\boldsymbol{x}_i - \boldsymbol{x})\, y_i}{\displaystyle\sum_{i=1}^{P} K_{\boldsymbol{H}}(\boldsymbol{x}_i - \boldsymbol{x})}, \tag{3.12}$$

which is the so-called *Nadaraya-Watson* estimator (NWE) [23], which is nothing but a space-varying convolution (if samples are irregularly spaced).

What we described above is the "classic" kernel regression framework, which as we just mentioned, yields a pointwise estimator that is always a local "linear", though possibly space-varying, combination of the neighboring samples. As such, it suffers from an inherent limitation. In the next sections, we describe the framework of *steering* KR in two and three dimensions, in which the kernel weights themselves are computed from the local window (or cube), and therefore we arrive at filters with more complex (nonlinear and space-varying) action on the data.

3.2.2 Steering Kernel Regression in 2-D

The steering kernel approach is based on the idea of robustly obtaining local signal structures by analyzing the radiometric (pixel value) differences locally, and feeding this structure information to the kernel function in order to affect its shape and size.

Consider the (2×2) smoothing matrix $\boldsymbol{H}$ in (3.5). As explained in Section 3.2.1, in the generic "classical" case, this matrix is a scalar multiple of the identity with the global parameter h. This results in kernel weights which have equal effect along the x_1- and x_2-directions. However, if we properly choose this matrix, the kernel function can capture local structures. More precisely, we define the smoothing matrix as a symmetric positive-definite matrix:

$$\boldsymbol{H}_i = h\boldsymbol{C}_i^{-\frac{1}{2}} \tag{3.13}$$

which we call the *steering matrix* and where, *for each given sample* y_i, the matrix $\boldsymbol{C}_i$ is estimated as the local covariance matrix of the neighborhood spatial gradient vectors. A naive estimate of this covariance matrix may be obtained as

$$\widehat{\boldsymbol{C}}_i^{\text{naive}} = \boldsymbol{J}_i^T \boldsymbol{J}_i, \tag{3.14}$$

with

$$\boldsymbol{J}_i = \begin{bmatrix} \vdots & \vdots \\ z_{x_1}(\boldsymbol{x}_j), & z_{x_2}(\boldsymbol{x}_j) \\ \vdots & \vdots \end{bmatrix}, \quad \boldsymbol{x}_j \in \xi_i, \quad j = 1, \cdots, Q, \tag{3.15}$$

where $z_{x_1}(\,\cdot\,)$ and $z_{x_2}(\,\cdot\,)$ are the first derivatives along x_1- and x_2-axes, ξ_i is the local analysis window around a sample position $\boldsymbol{x}_i$, and Q is the number of rows in $\boldsymbol{J}_i$. However, the naive estimate may in general be rank deficient or unstable. Therefore, instead of using the naive estimate, we obtain covariance matrices by using the (compact) singular value decomposition (SVD) of $\boldsymbol{J}_i$. A specific choice of $\boldsymbol{C}_i$ using the SVD for the 2-D case is introduced in [31], and we will show $\boldsymbol{C}_i$ for the 3-D case in Section 3.2.3.

With the above choice of the smoothing matrix and a Gaussian kernel, we now have the steering kernel function as

$$K_{\boldsymbol{H}_i}(\boldsymbol{x}_i - \boldsymbol{x}) = \frac{\sqrt{\det(\boldsymbol{C}_i)}}{2\pi h^2} \exp\left\{-\frac{1}{2h^2}\left\|\boldsymbol{C}_i^{\frac{1}{2}}(\boldsymbol{x}_i - \boldsymbol{x})\right\|_2^2\right\}, \tag{3.16}$$

and the weighted least square estimator as

$$\widehat{\boldsymbol{b}} = \left(\boldsymbol{X}^T \boldsymbol{K}^{\text{s}} \boldsymbol{X}\right)^{-1} \boldsymbol{X}^T \boldsymbol{K}^{\text{s}}\, \boldsymbol{y} \tag{3.17}$$

where

$$\boldsymbol{K}^{\text{s}} = \text{diag}\left[\; K_{\boldsymbol{H}_1}(\boldsymbol{x}_1 - \boldsymbol{x}), \quad K_{\boldsymbol{H}_2}(\boldsymbol{x}_2 - \boldsymbol{x}), \quad \cdots, \quad K_{\boldsymbol{H}_P}(\boldsymbol{x}_P - \boldsymbol{x}) \;\right]. \tag{3.18}$$

Again, for example, for zeroth order (i.e. $N = 0$), the estimator (3.17) yields a pointwise estimator:

$$\hat{z}(\boldsymbol{x}) = \widehat{\beta}_0 = \frac{\sum_{i=1}^{P} K_{\boldsymbol{H}_i}(\boldsymbol{x}_i - \boldsymbol{x})\, y_i}{\sum_{i=1}^{P} K_{\boldsymbol{H}_i}(\boldsymbol{x}_i - \boldsymbol{x})}, \tag{3.19}$$

which is the data-adapted version of NWE. It is noteworthy that, as shown in the weight matrix $\boldsymbol{K}^{\mathrm{s}}$ (3.18) involving the steering matrices $\{\boldsymbol{H}_i\}_{i=1}^{P}$ of all the neighboring samples $\{y_i\}_{i=1}^{P}$, the steering kernel function (3.16) effectively captures the local image structures. We will graphically show the steering kernels shortly in Figure 3.3.

Figure 3.2 illustrates a schematic representation of the estimates of local covariance matrices $\boldsymbol{C}_i$ in (3.13) at a local region with one dominant orientation. First, we compute gradients $z_{x_1}(\,\cdot\,)$ and $z_{x_2}(\,\cdot\,)$ of the given image, where the gradients are illustrated as vectors with red arrows. In this example, we set the size of the regression window ω to 5×5 and and the size of the window ξ for the calculation of the covariance estimate is set to 3×3. Therefore, the overall analysis window becomes 7×7. Next, sliding the window ξ_i for $i = 1, \cdots, 25$, we compute the covariance matrix $\boldsymbol{C}_i$ for each pixel y_i in the middle (5×5) portion. Once $\boldsymbol{C}_i$'s are available, we perform the steering kernel regression (3.17) with the weights given by the $\boldsymbol{C}_i$'s and estimate the pixel value $z(\boldsymbol{x})$ at the position of interest. Graphical representations of the steering kernel weights for noise-free (Pepper and Parrot) images are illustrated in

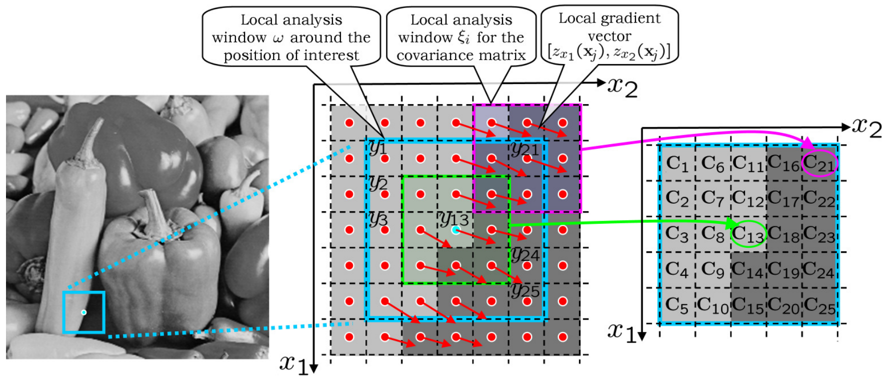

Local analysis window ω around the position of interest
Local analysis window ξ_i for the covariance matrix
Local gradient vector $[z_{x_1}(\mathbf{x}_j), z_{x_2}(\mathbf{x}_j)]$
x_2
x_1
y_1
y_2
y_3
y_{13}
y_{21}
y_{24}
y_{25}
x_2
x_1
C_1 C_6 C_{11} C_{16} C_{21}
C_2 C_7 C_{12} C_{17} C_{22}
C_3 C_8 C_{13} C_{18} C_{23}
C_4 C_9 C_{14} C_{19} C_{24}
C_5 C_{10} C_{15} C_{20} C_{25}

FIGURE 3.2: A schematic representation of the estimates of local covariance matrices at a local region with one dominant orientation: First, we estimate the gradients and compute the local covariance matrix $\boldsymbol{C}_i$ from the local gradient vectors for each pixel in the local analysis window ω_i around the position of interest (i.e. $\boldsymbol{x}_{13}$ in the figure).

Figure 3.3. Figures 3.3(c) and (d) show the steering weight $\boldsymbol{K}^{\mathrm{s}}$ given by (3.16) without $\sqrt{\det \boldsymbol{C}_i}/2\pi h^2$ at every 11 pixels in the horizontal and vertical directions. It should be noted that we compute the steering kernels $K_{\boldsymbol{H}_i}(\boldsymbol{x}_i - \boldsymbol{x})$ as a function of each $\boldsymbol{x}_i$ with the position of interest $\boldsymbol{x}$ held fixed. Thus, the kernel involves not only $\boldsymbol{C}_i$ at the position of interest but also its neighborhoods', and the steering kernel weights effectively take local image structures into account. Moreover, the steering weights spread wider in flat regions and spread along edges while staying small at the texture regions (for example the region around Parrot's eye). Therefore, the steering kernel filtering smoothes pixels strongly along the local structures rather than across them. Figures 3.3(e) and (f) show the scalar values $\sqrt{\det \boldsymbol{C}_i}/2\pi h^2$ of (3.16). The scalars become large at edges and textured regions and small at flat regions. We also note that even for the highly noisy case, we can obtain stable estimates of local structure [31].

3.2.3 Space-time (3-D) Steering Kernel Regression

So far, we presented the KR framework in 2-D. In this section, we introduce the time axis and present *space-time* SKR to process video data. As mentioned in the introductory section, we explain how this extension can yield a remarkable advantage in that space-time SKR does not necessitate explicit (sub-pixel) motion estimation.

First, introducing the time axis, similar to the 2-D data model, we have the data model in 3-D as

$$y_i = z(\boldsymbol{x}_i) + \varepsilon_i, \quad \boldsymbol{x}_i \in \omega, \quad i = 1, \cdots, P, \tag{3.20}$$

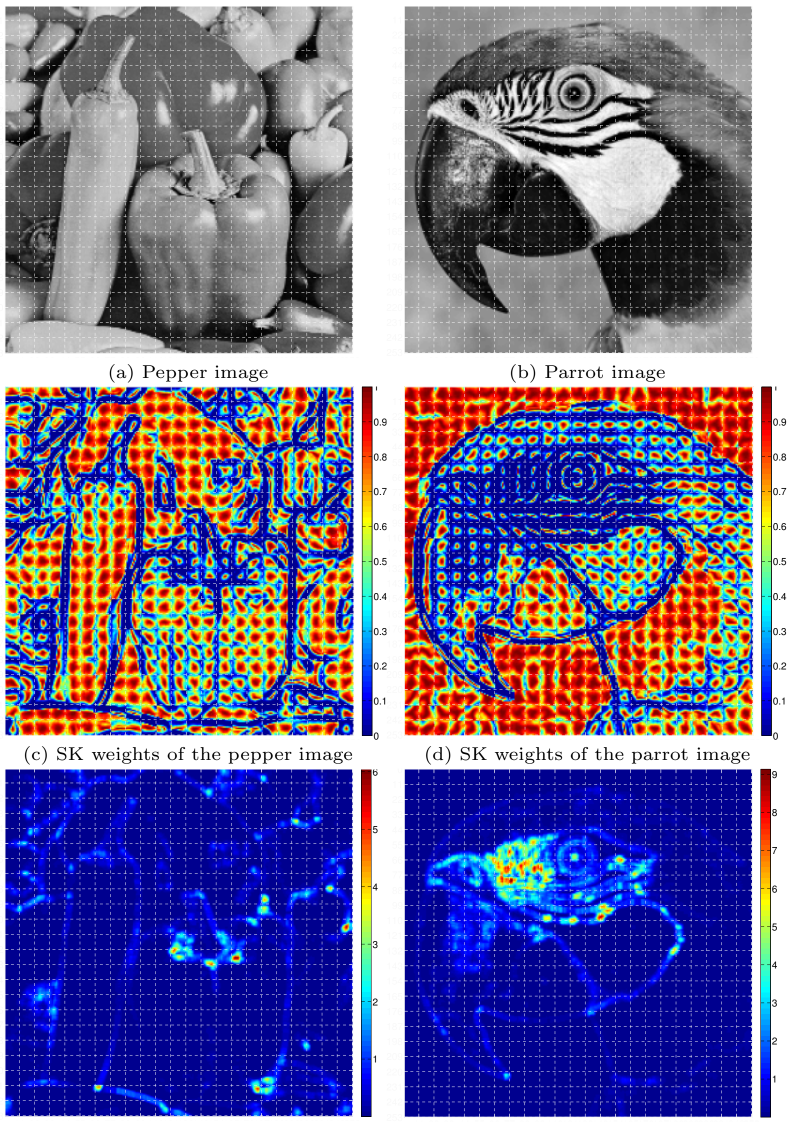
where y_i is a noise-ridden sample at $\boldsymbol{x}_i = [x_{1i}, x_{2i}, t_i]^T$, x_{1i} and x_{2i} are spatial coordinates, t_i $(= x_{3i})$ is the temporal coordinate, $z(\,\cdot\,)$ is the regression function of interest, ε_i is an *i.i.d.* zero-mean noise process, and P is the total number of nearby samples in a 3-D neighborhood ω of interest, which we will henceforth call ω a "cubicle". As in (3.2), we also locally approximate $z(\,\cdot\,)$ by a Taylor series in 3-D, where $\boldsymbol{\nabla}$ and $\boldsymbol{\mathcal{H}}$ are now the gradient (3×1) and Hessian (3×3) operators, respectively. With a (3×3) steering matrix $(\boldsymbol{H}_i)$, the estimator takes the familiar form:

$$\hat{z}(\boldsymbol{x}) = \widehat{\beta}_0 = \sum_{i=1}^{P} W_i(K, \boldsymbol{H}_i, N, \boldsymbol{x}_i - \boldsymbol{x})\, y_i. \tag{3.21}$$

It is worth noting that 3-D SKR is a pointwise estimator of the regression function $z(\,\cdot\,)$ and it is capable of estimating a pixel value at arbitrary space-time positions $\boldsymbol{x}$. The derivation for the steering matrix is quite similar to the 2-D case. Indeed, we again define $\boldsymbol{H}_i$ as

$$\boldsymbol{H}_i = h\boldsymbol{C}_i^{-\frac{1}{2}}, \tag{3.22}$$

where the covariance matrix $\boldsymbol{C}_i$ can be naively estimated as $\widehat{\boldsymbol{C}}_i^{\text{naive}} = \boldsymbol{J}_i^T \boldsymbol{J}_i$



(a) Pepper image (b) Parrot image

(c) SK weights of the pepper image (d) SK weights of the parrot image

(e) Scaling values of the pepper image (f) Scaling values of the parrot image

FIGURE 3.3: Graphical representations of steering kernel weights (3.18) for (a) Pepper and (b) Parrot images: The figures (c) and (d) illustrate the steering weight matrices $\boldsymbol{K}^{\text{s}}$ given by (3.16) without $\sqrt{\det \boldsymbol{C}_i}/2\pi h^2$ at every 11 pixels in horizontal and vertical directions. For this illustration, we chose the analysis window sizes $\omega = 11 \times 11$ and $\xi = 5 \times 5$. The figures (e) and (f) shows the scalar values $\sqrt{\det \boldsymbol{C}_i}/2\pi h^2$. The scalars becomes large at edge and textured regions and small at flat regions.

with

$$\boldsymbol{J}_i = \begin{bmatrix} \vdots & \vdots & \vdots \\ z_{x_1}(\boldsymbol{x}_j), & z_{x_2}(\boldsymbol{x}_j), & z_t(\boldsymbol{x}_j) \\ \vdots & \vdots & \vdots \end{bmatrix}, \quad \boldsymbol{x}_j \in \xi_i, \quad j = 1, \cdots, Q, \tag{3.23}$$

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 $\boldsymbol{x}_i$, and Q is the number of rows in $\boldsymbol{J}_i$. Once again for the sake of robustness, as explained in Section 3.2.2, we compute a more stable estimate of $\boldsymbol{C}_i$ by invoking the SVD of $\boldsymbol{J}_i$ with regularization as:

$$\widehat{\boldsymbol{C}}_i = \gamma_i \sum_{q=1}^{3} \varrho_q \boldsymbol{v}_q \boldsymbol{v}_q^T, \tag{3.24}$$

with

$$\begin{aligned} \varrho_1 &= \frac{s_1 + \lambda'}{s_2 s_3 + \lambda'}, \quad \varrho_2 = \frac{s_2 + \lambda'}{s_1 s_3 + \lambda'}, \\ \varrho_3 &= \frac{s_3 + \lambda'}{s_1 s_2 + \lambda'}, \quad \gamma_i = \left(\frac{s_1 s_2 s_3 + \lambda''}{Q} \right)^{\alpha}, \end{aligned} \tag{3.25}$$

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 3.5.1 for the derivations), and Q is the number of rows in $\boldsymbol{J}_i$. 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 ($\boldsymbol{v}_1$, $\boldsymbol{v}_2$, and $\boldsymbol{v}_3$) are given by the (compact) SVD of $\boldsymbol{J}_i$:

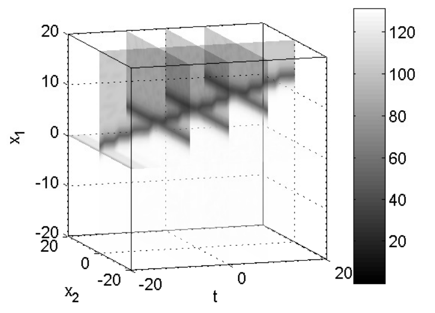
$$\boldsymbol{J}_i = \boldsymbol{U}_i \boldsymbol{S}_i \boldsymbol{V}_i^T = \boldsymbol{U}_i \ \mathrm{diag}\left\{s_1, s_2, s_3\right\} [\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3]^T. \tag{3.26}$$

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

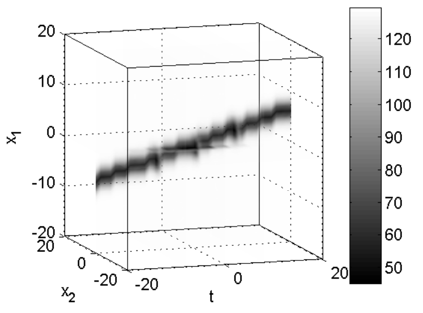
$$K_{\boldsymbol{H}_i}(\boldsymbol{x}_i - \boldsymbol{x}) = \sqrt{\frac{\det(\boldsymbol{C}_i)}{(2\pi h^2)^3}} \exp\left\{ -\frac{1}{2h^2} \left\| \boldsymbol{C}_i^{\frac{1}{2}} (\boldsymbol{x}_i - \boldsymbol{x}) \right\|_2^2 \right\}, \tag{3.27}$$

with $\boldsymbol{x} = [x_1, x_2, t]$. The main tuning parameters are the global smoothing parameter (h) in (3.27) and the structure sensitivity (α) in (3.25). The specific choices of these parameters are indicated in Section 3.3, and Appendix 3.5.2 gives more details about the parameters h and α.

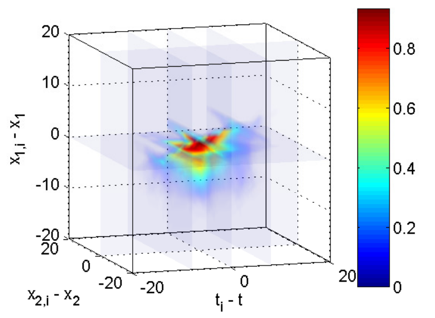
Figure 3.4 shows visualizations of the 3-D weights given by the steering kernel function for two cases: (a) a horizontal edge moving vertically over time (creating a tilted plane in the local cubicle), and (b) a small circular dot also moving vertically over time (creating a thing tube in a local cubicle). Considering the case of denoising for the pixel located at the center of each data cube of Figures 3.4(a) and (b), we have the steering kernel weights illustrated in Figures 3.4(c)(d) and (e)(f). Figures 3.4(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.



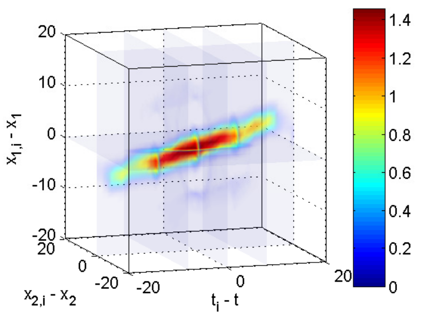
(a) A tilted plane



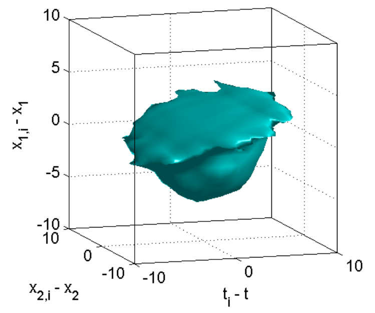
(b) A thin tube



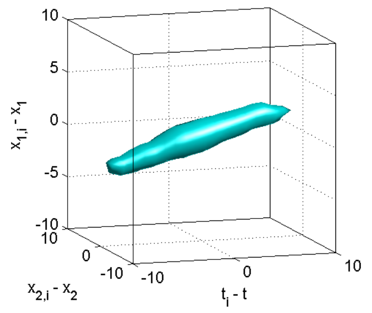
(c) Cross-sections of SK weights of (a)



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



(e) The isosurface of (c)



(f) The isosurface of (d)

FIGURE 3.4: 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.

Also, Figure 3.5 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 $\xi_i = 5\times5\times5$. 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 3.4 and 3.5, 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 [1, 11, 20] to describe the underlying idea. Defining the 3-D motion vector as $\widetilde{\boldsymbol{m}}_i = [m_1, m_2, 1]^T = [\boldsymbol{m}_i^T, 1]^T$ and invoking the brightness constancy equation (BCE) [15] in a local cubicle centered at $\boldsymbol{x}_i$, we can use the matrix of

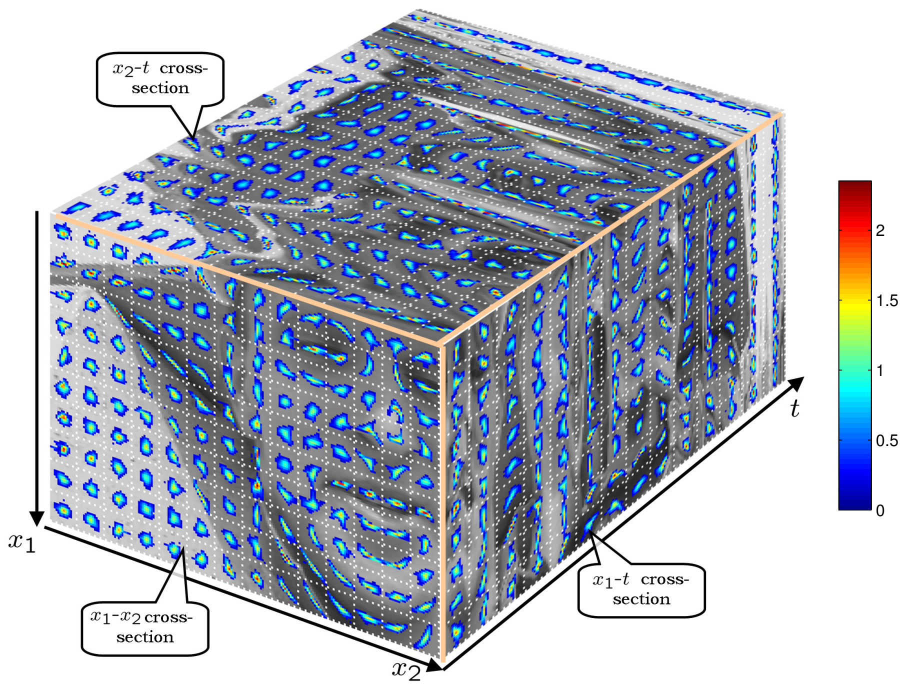


FIGURE 3.5: A graphical representation of 3-D steering kernel weights (3.27) for the Foreman sequence: The figure illustrate cross-sections of the steering weight matrices $\boldsymbol{K}^{\mathrm{s}}$ given by (3.27) at every 15 pixels in horizontal, vertical, and time. For the illustration, we chose the analysis cubicle sizes $\omega = 15 \times 15 \times 15$ and $\xi_i = 5 \times 5 \times 5$.

gradients $\boldsymbol{J}_i$ in (3.23) to write the BCE as

$$\boldsymbol{J}_i \widetilde{\boldsymbol{m}}_i = \boldsymbol{J}_i \begin{bmatrix} \boldsymbol{m}_i \\ 1 \end{bmatrix} = \underline{\mathbf{0}}. \tag{3.28}$$

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

$$\boldsymbol{J}_i^T \boldsymbol{J}_i \widetilde{\boldsymbol{m}}_i = \widehat{\boldsymbol{C}}_i^{\text{naive}} \widetilde{\boldsymbol{m}}_i \approx \underline{\mathbf{0}}. \tag{3.29}$$

Now invoking the decomposition of $\widehat{\boldsymbol{C}}_i$ in (3.24), we can write

$$\sum_{q=1}^{3} \varrho_q \boldsymbol{v}_q \left(\boldsymbol{v}_q^T \widetilde{\boldsymbol{m}}_i \right) \approx \underline{\mathbf{0}}. \tag{3.30}$$

The above decomposition shows explicitly the relationship between the motion vector and the principal orientation directions computed with in the SKR framework. The most generic scenario in a small cubicle is one where the local texture are 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 $\boldsymbol{v}_3$ (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 $\frac{\widetilde{\boldsymbol{m}}_i}{\|\widetilde{\boldsymbol{m}}_i\|}$ [40, 3]. 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 (3.25). 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 our other work in [34], we presented the *motion-assisted steering kernel* (MASK) method, which explicitly feeds 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 3.5, 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.

3.2.4 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[2]. In Figures 3.6(a) and (b), we illustrate the contours of steering kernels the pixel of interest located at the center of the middle frame. For the small displacement case illustrated in Figure 3.6(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 3.6(b), similar pixels, though close in the time 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 "neutralize") for it, as illustrated in Figure 3.6(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 3.7(b),

[2]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.

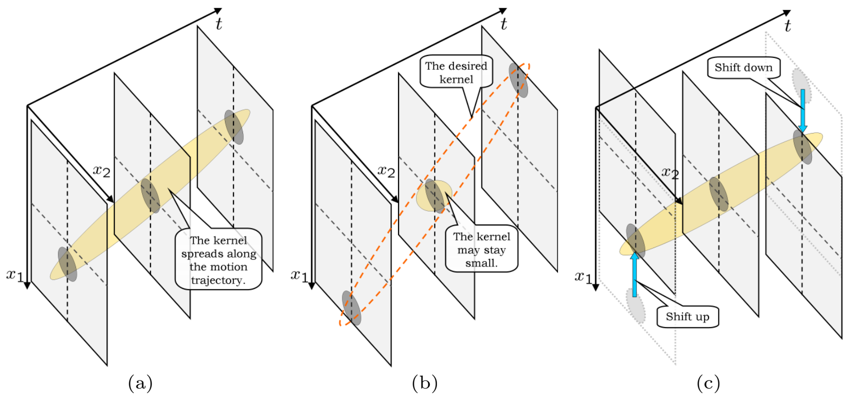

t
x_2
x_1
The kernel spreads along the motion trajectory.
(a)
The desired kernel
The kernel may stay small.
(b)
Shift down
Shift up
(c)

FIGURE 3.6: 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.

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 importance 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.

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 ($\boldsymbol{m}_i^{\text{large}}$) and small but complex motion ($\boldsymbol{m}_i$):

$$\boldsymbol{m}_i^{\text{true}} = \boldsymbol{m}_i^{\text{large}} + \boldsymbol{m}_i, \tag{3.31}$$

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

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

$$\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i + \begin{bmatrix} \boldsymbol{m}_i^{\text{large}} \\ 0 \end{bmatrix} (t_i - t), \tag{3.32}$$

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 $y(\tilde{\boldsymbol{x}}_i)$ does not require any interpolation, and therefore no artifacts are introduced. Rewriting the 3-D SKR problem for the new

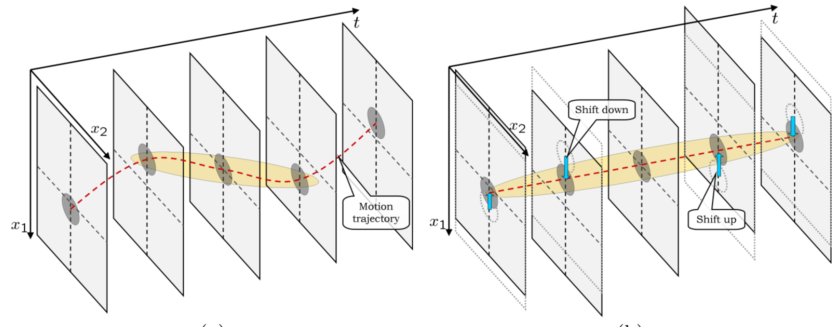


FIGURE 3.7: Steering kernel footprints for (a) a video with a complex motion trajectory, and (b) the video after neutralizing the relatively large displacements.

sequence $y(\tilde{\boldsymbol{x}}_i)$, we have:

$$\min_{\{\boldsymbol{\beta}_n\}_{n=0}^{N}} \sum_{i=1}^{P} \left[y(\tilde{\boldsymbol{x}}_i) - \beta_0 - \boldsymbol{\beta}_1^T(\tilde{\boldsymbol{x}}_i - \boldsymbol{x}) - \boldsymbol{\beta}_2^T \operatorname{vech}\left\{(\tilde{\boldsymbol{x}}_i - \boldsymbol{x})(\tilde{\boldsymbol{x}}_i - \boldsymbol{x})^T\right\} - \cdots \right]^2 \cdot K_{\widetilde{\boldsymbol{H}}_i}(\tilde{\boldsymbol{x}}_i - \boldsymbol{x}) \tag{3.33}$$

where the steering matrix $\widetilde{\boldsymbol{H}}_i$ is computed from the motion-compensated sequence $y(\tilde{\boldsymbol{x}}_i)$. Similar to the 2-D estimator (3.11), the above minimization yields the following pixel estimator at the position of interest ($\boldsymbol{x}$) as

$$\begin{aligned} \hat{z}(\boldsymbol{x}) &= \widehat{\beta}_0 = \boldsymbol{e}_1^T \left(\widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{K}}^{\mathrm{s}} \widetilde{\boldsymbol{X}} \right)^{-1} \widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{K}}^{\mathrm{s}}\, \tilde{\boldsymbol{y}} \\ &= \sum_{i=1}^{P} W_i(K, \widetilde{\boldsymbol{H}}_i, N, \tilde{\boldsymbol{x}}_i - \boldsymbol{x})\, y(\tilde{\boldsymbol{x}}_i), \end{aligned} \tag{3.34}$$

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

$$\tilde{\boldsymbol{y}} = \left[\ y(\tilde{\boldsymbol{x}}_1),\quad y(\tilde{\boldsymbol{x}}_2),\quad \cdots,\quad y(\tilde{\boldsymbol{x}}_P)\ \right]^T, \quad \boldsymbol{b} = \left[\ \beta_0,\quad \boldsymbol{\beta}_1^T,\quad \cdots,\quad \boldsymbol{\beta}_N^T\ \right]^T,$$

$$\widetilde{\boldsymbol{K}}^{\mathrm{s}} = \operatorname{diag}\left[\ K_{\widetilde{\boldsymbol{H}}_1}(\tilde{\boldsymbol{x}}_1 - \boldsymbol{x}),\quad K_{\widetilde{\boldsymbol{H}}_2}(\tilde{\boldsymbol{x}}_2 - \boldsymbol{x}),\quad \cdots,\quad K_{\widetilde{\boldsymbol{H}}_P}(\tilde{\boldsymbol{x}}_P - \boldsymbol{x})\ \right], \tag{3.35}$$

and

$$\widetilde{\boldsymbol{X}} = \begin{bmatrix} 1, & (\tilde{\boldsymbol{x}}_1 - \boldsymbol{x}), & \operatorname{vech}\left\{(\tilde{\boldsymbol{x}}_1 - \boldsymbol{x})(\tilde{\boldsymbol{x}}_1 - \boldsymbol{x})^T\right\}, & \cdots \\ 1, & (\tilde{\boldsymbol{x}}_2 - \boldsymbol{x}), & \operatorname{vech}\left\{(\tilde{\boldsymbol{x}}_2 - \boldsymbol{x})(\tilde{\boldsymbol{x}}_2 - \boldsymbol{x})^T\right\}, & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ 1, & (\tilde{\boldsymbol{x}}_P - \boldsymbol{x}), & \operatorname{vech}\left\{(\tilde{\boldsymbol{x}}_P - \boldsymbol{x})(\tilde{\boldsymbol{x}}_P - \boldsymbol{x})^T\right\}, & \cdots \end{bmatrix}. \tag{3.36}$$

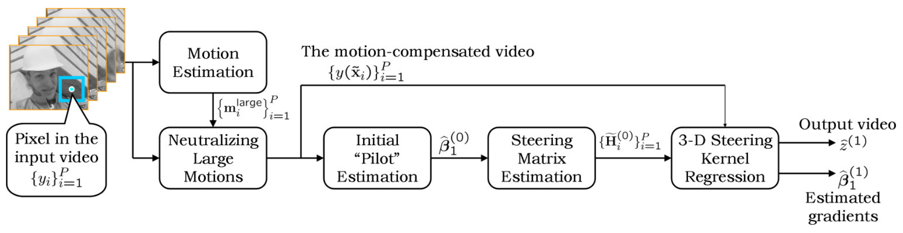
In the following section, we further elaborate on the implementation of the 3-D SKR for enhancement and super-resolution, including its iterative application.

3.2.5 Implementation and Iterative Refinement

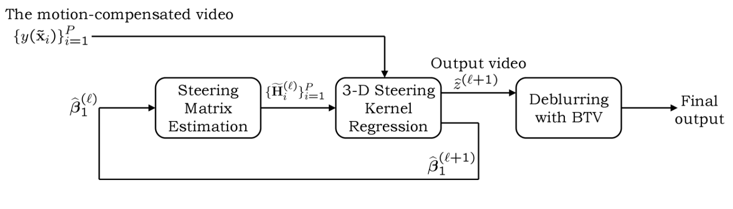
As we explained earlier, since the performance of the SKR depends on the accuracy of the orientations, we refine it to derive an iterative algorithm we call *iterative SKR* (ISKR), which results in improved orientation estimats and therefore a better final denoising and upscaling result. The extension 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 3.8 is as follow.

First, estimate the large motions ($\boldsymbol{m}_i^{\text{large}}$) of the given input sequence ($\{y_i\}_{i=1}^P$). Then using $\boldsymbol{m}_i^{\text{large}}$, we neutralize the large motions and generate a motion-compensated video sequence ($\{y(\tilde{\boldsymbol{x}}_i)\}_{i=1}^P$). 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 $\{\tilde{\boldsymbol{x}}_i\}_{i=1}^P$ of the motion-compensated video. This process is indicated as the "pilot" estimate in the block diagram. After that, we create steering matrices ($\widetilde{\boldsymbol{H}}_i^{(0)}$) for all the samples $y(\tilde{\boldsymbol{x}}_i)$ by (3.22) and (3.24). Once $\widetilde{\boldsymbol{H}}_i^{(0)}$ are available, we plug them into the kernel weight matrix (3.35) and estimate not only an unknown pixel value ($z(\boldsymbol{x})$) at a position of interest ($\boldsymbol{x}$) by (3.34) but also its gradients ($\widehat{\boldsymbol{\beta}}_1^{(1)}$). This is the initialization process shown in Figure 3.8(a). Next, using $\widehat{\boldsymbol{\beta}}_1^{(1)}$, we re-create the steering matrices $\widetilde{\boldsymbol{H}}_i^{(1)}$. 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 $\widetilde{\boldsymbol{H}}_i^{(1)}$, we apply SKR to the embedded input video again. We repeat this process several times as shown in Figure 3.8(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 are necessary to reach convergence[3]. Finally, we perform deblurring on the upscaled videos

[3]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,


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

(a) Initialization


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

(b) Iteration

FIGURE 3.8: Block diagram representation of the 3-D iterative steering kernel regression with motion compensation: (a) the initialization process, and (b) the iteration process.

to recover the high frequency components. Appendix 3.5.3 shows the detailed method that we use in this work.

Figure 3.9 illustrates a simple super-resolution example, where we created 9 of synthetic low resolution frames from the image shown in Figure 3.9(a) by blurring with a 3×3 uniform PSF, shifting the blurred image by 0, 4, or 8 pixels[4] 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 Figure 3.9(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

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) [33]

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

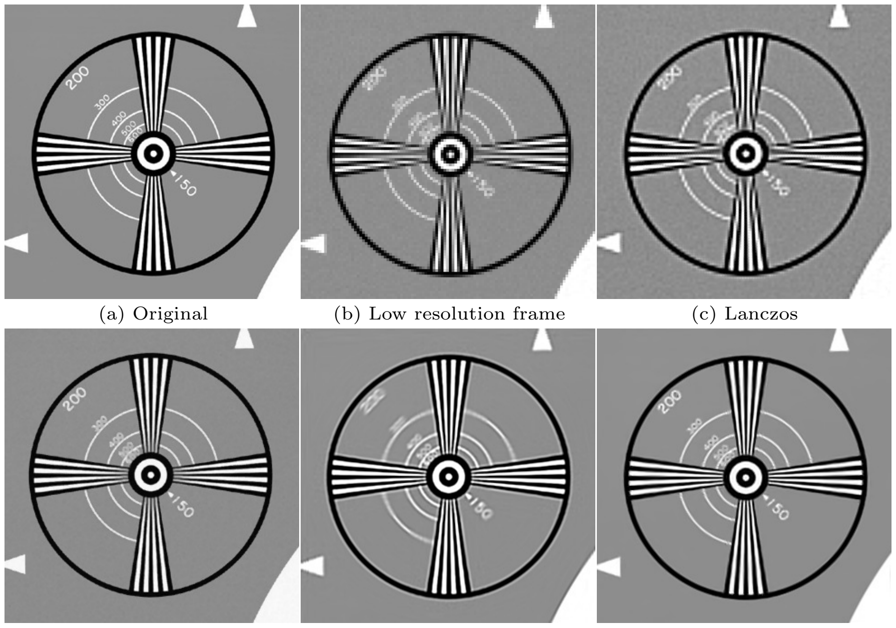


FIGURE 3.9: A simple super-resolution example: (a) the original image, (b) one of 9 low resolution images generated by blurring with a 3×3 uniform PSF, spatially downsampling with a factor of $3:1$, and adding white Gaussian noise with standard deviation 2, (c) an upscaled image by Lanczos (single frame upscale), (d) an upscaled image by robust super-resolution (SR) [8], and (e) an upscaled image by non-local mean (NLM) based super-resolution [28], 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.

steering kernel weights cannot spread effectively along time as illustrated in Figure 3.7(a). The upscaled frames by Lanczos, robust super-resolution [8], non-local mean based super-resolution [28], and 3-D ISKR with rough motion estimation at time $t = 5$ are shown in Figures 3.9(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 3.9.

It is worth noting that while in the proposed algorithm in Figure 3.8, 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 [7]. Similarly, in our implementation, we used the optical flow framework [21] to compute the rough motion estimates. This step too can be replaced by other methods such as a block matching algorithm [41].

3.3 Examples

The utility and novelty of our algorithm lies in the fact that it is 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.

3.3.1 Spatial Upscaling Examples

In this section, we present some denoising/upscaling examples. The sequences in this section contain motions of relatively modest size due to the effect of severe spatial downsampling (we downsampled original videos spatially with the downsampling factor $3 : 1$) and therefore motion compensation as we described earlier was not necessary. In Section 3.3.2, we illustrate additional examples of spatiotemporal video upscaling.

First, we degrade two videos (Miss America and Foreman sequences), using the first 30 frames of each sequence, blurring with a 3×3 uniform point spread function (PSF), spatially downsampling the videos by a factor of $3 : 1$ in the horizontal and vertical directions, and then adding white Gaussian noise with standard deviation $\sigma = 2$. Two of the selected degraded frame at time $t = 14$ for Miss America and $t = 7$ for Foreman are shown in Figures 3.10(a) and 3.11(a), respectively. Then, we simultaneously upscale and denoise the degraded videos by Lanczos interpolation (frame-by-frame upscaling), the NL-

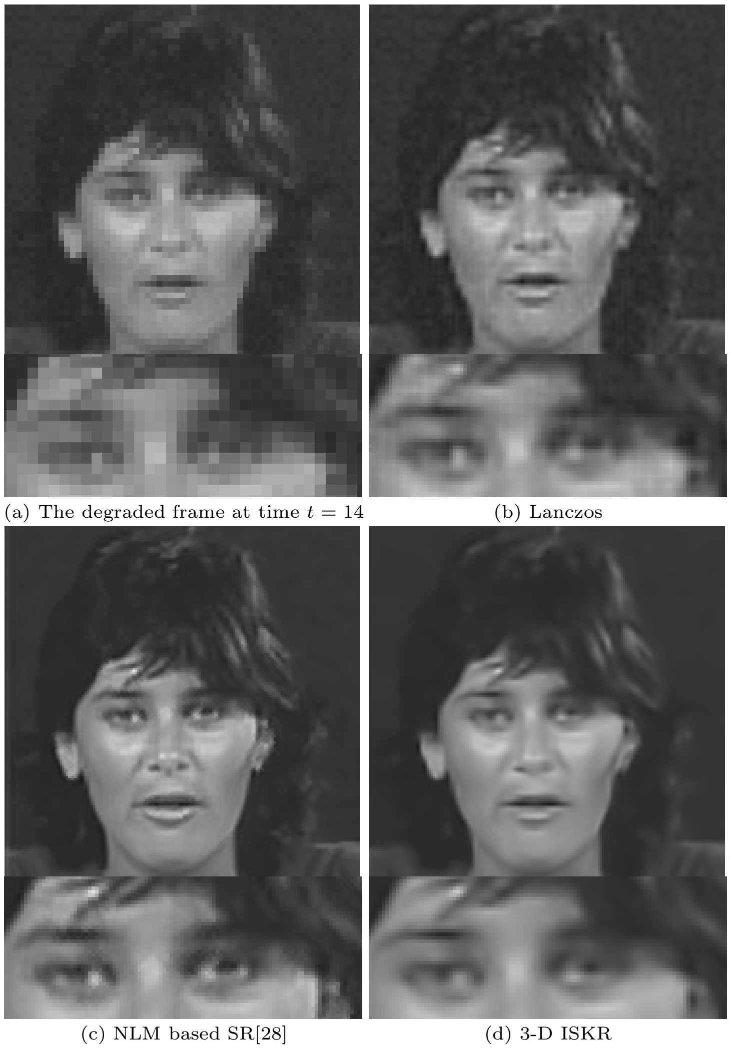


FIGURE 3.10: A video upscale example using Miss America sequence: (a) the degraded frame at time $t = 14$, (b) the upscaled frame by Lanczos interpolation (PSNR = 34.25[dB]), (c) the upscaled frame by NLM-means based SR [28] (PSNR = 34.95[dB]), and (d) the upscaled frame by 3-D ISKR (PSNR = 35.65[dB]). Also, the PSNR values for all the frames are shown in Figure 3.12(a).

means based approach of [28], and 3-D ISKR, which includes deblurring[5] the upscaled video frames using the BTV approach [8]. Hence, we used a radially symmetric Gaussian PSF which reflects an "average" PSF induced by the kernel function used in the reconstruction process. The final upscaled results are shown in Figures 3.10(b)-(d) and 3.11(b)-(d), respectively. The corresponding average PSNR values across all the frames for the Miss America example

[5]Note that the 3×3 uniform PSF is no longer suitable for the deblurring since the kernel regression gives its own blurring effects.

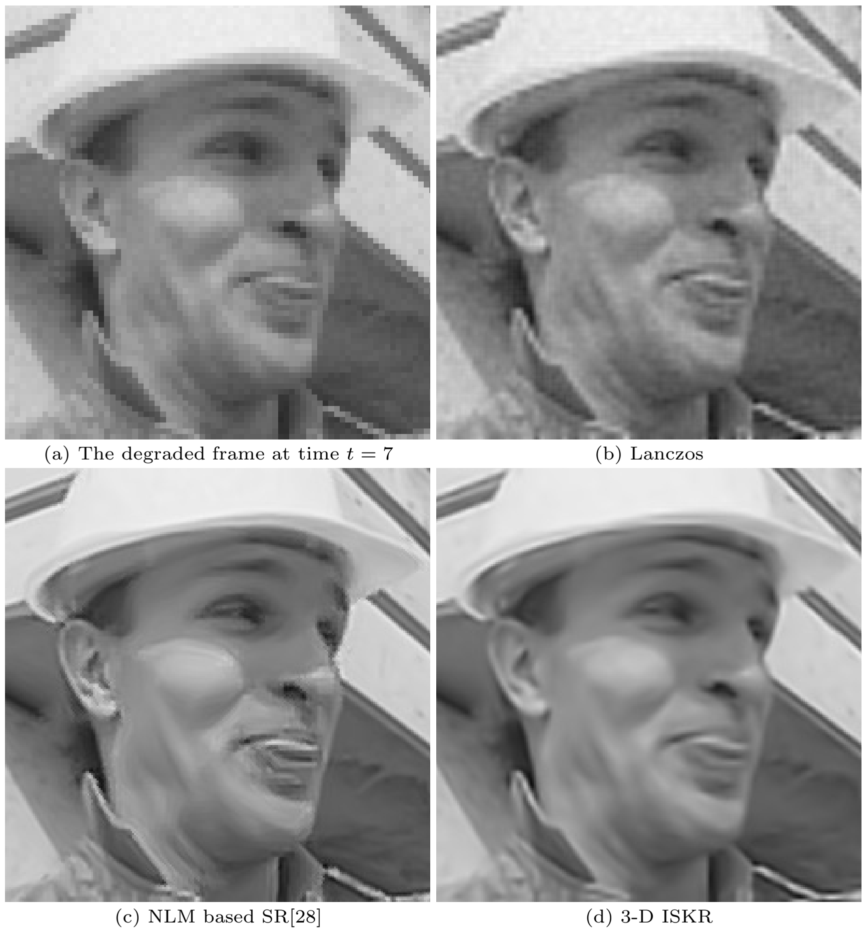

(a) The degraded frame at time $t = 7$ (b) Lanczos
(c) NLM based SR[28] (d) 3-D ISKR

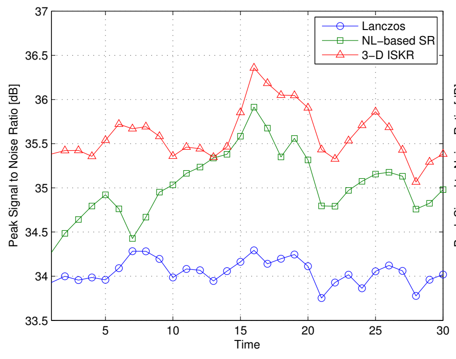
FIGURE 3.11: A video upscaling example using Foreman sequence: (a) the degraded frame at time $t = 7$, (b) the upscaled frame by Lanczos interpolation (PSNR = 30.98[dB]), (c) the upscaled frame by NL-means based SR [28] (PSNR = 32.21[dB]), and (d) the upscaled frame by 3-D ISKR (PSNR = 33.58[dB]). Also the PSNR values for all the frames are shown in Figure 3.12(b)

are 34.05[dB] (Lanczos), 35.04[dB] (NL-means based SR [28]), and 35.60[dB] (3-D ISKR) and the average PSNR values for Foreman are 30.43[dB] (Lanczos), 31.87[dB] (NL-means based SR), and 32.60[dB] (3-D ISKR), respectively. The graphs in Figure 3.12 illustrate the PSNR values frame by frame. It is interesting to note that while the NL-means method appears to produce more crisp results in this case, the corresponding PSNR values for this method are surprisingly lower than that for the proposed 3-D ISKR method. We believe, as partly indicated in Figure 3.14, that this may be in part due to some left-over high frequency artifacts and possibly lesser denoising capability of the NL-means method.

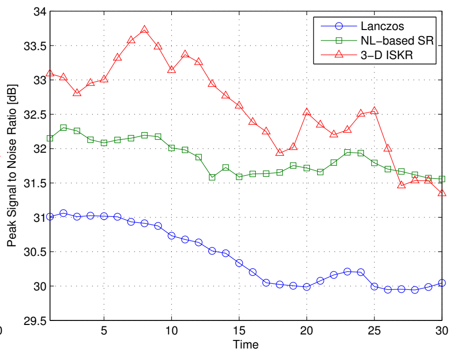
As for the parameters of our algorithm, we applied SKR with the global smoothing parameter $h = 1.5$, the local structure sensitivity $\alpha = 0.1$ and a $5 \times 5 \times 5$ local cubicle and used an 11×11 Gaussian PSF with a standard deviation of 1.3 for the deblurring of Miss America and Foreman sequences. For the experiments shown in Figures 3.10 and 3.11, we iterated SKR 6 times.

The next example is a spatial upscaling example using a section of a real HDTV video sequence (300×300 pixels, 24 frames), shown in Figure 3.13(a), where no additional simulated degradation is added. As seen in the input frames, the video has real compression artifacts (i.e. blocking). In this example, we show the deblocking capability of the proposed method, and the upscaled results by Lanczos interpolation, NLM-based SR [28] and 3-D ISKR with a factor of 1 : 3 (i.e. the output resolution is 900×900 pixels) are shown in Figures 3.13(b)-(d), respectively. The proposed method[6] is able to remove the blocking artifacts effectively as well as to upscale the video.

[6]We applied our method to the luminance channel only.



(a) Miss America

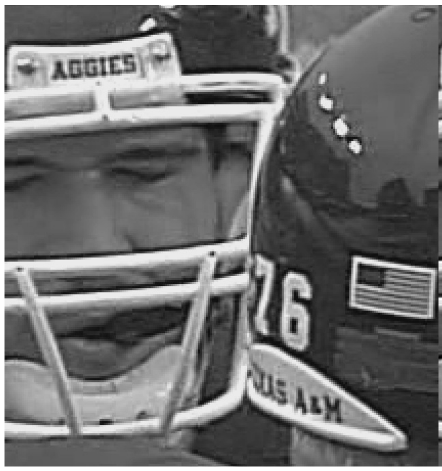


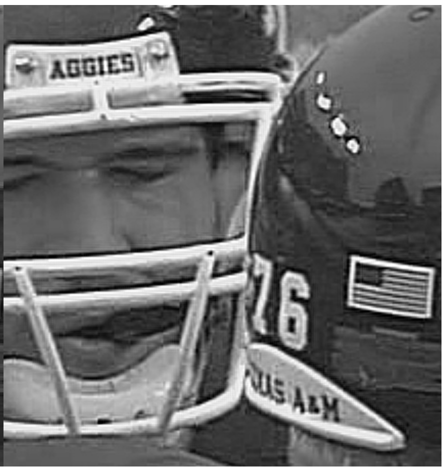
(b) Foreman

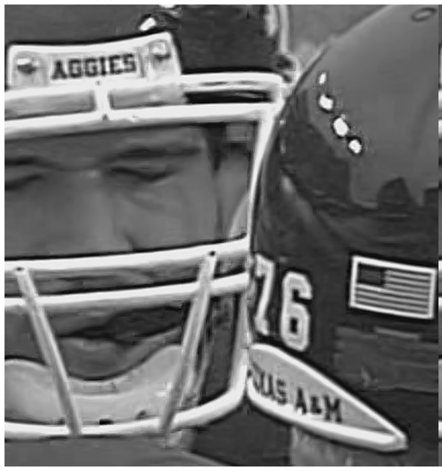
FIGURE 3.12: The PSNR values of each upscaled frame by Lanczos, NL-means based SR [28], and 3-D ISKR for (a) the results of Miss America shown in Fig. 3.10 and (b) the results of Foreman shown in Fig. 3.11.

3.3.2 Spatiotemporal Upscaling Examples

In this section, we present two spatiotemporal upscaling examples (also known as, frame interpolation and frame rate upconversion) by 3-D ISKR using Carphone and Salesman sequences. Unlike the previous examples (Miss America and Foreman), in the next examples, Carphone sequence has relatively large


(a) The input frame at $t = 5$


(b) Lanczos


(c) NLM-based SR[28]

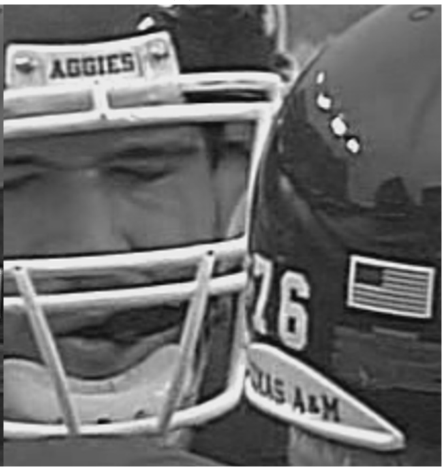
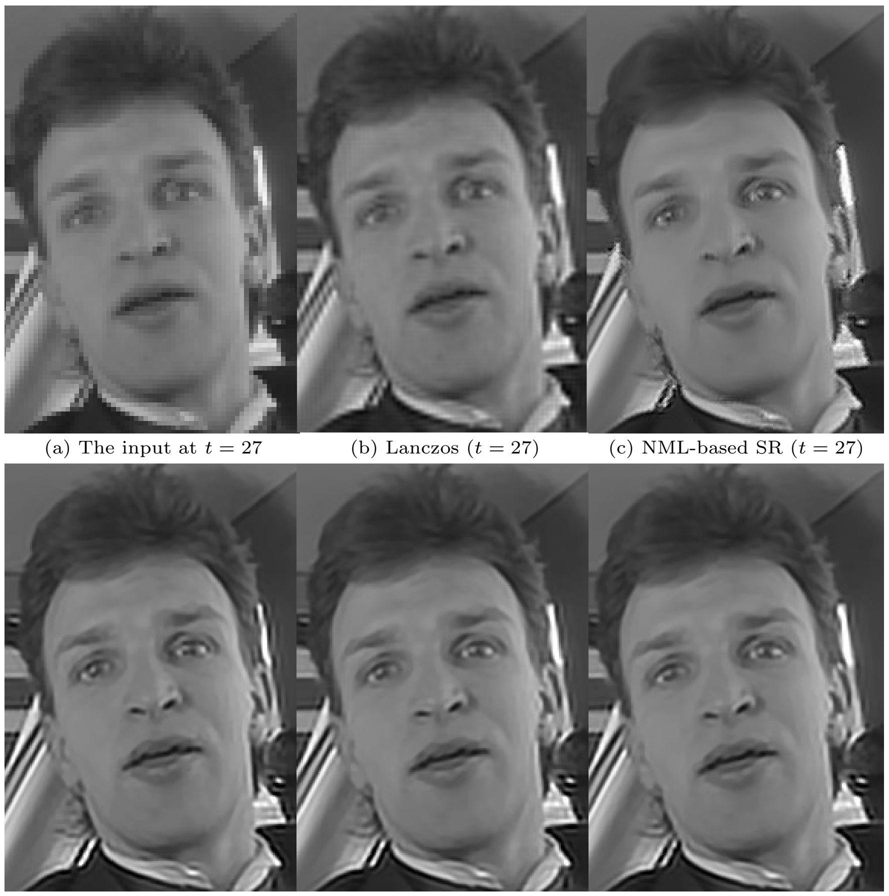

(d) 3-D ISKR

FIGURE 3.13: A spatial upscaling example of a real video: (a) Texas football sequence in luminance channel, and (b)-(d) the upscaled frames by Lanczos interpolation, NL-based SR [28] and 3-D ISKR, respectively. The input sequence has 24 frames in total and it is a real HD-TV content which carries compression artifacts, namely block artifacts. We upscale the video with the spatial upscaling factor of 1 : 3.

and more complex displacements between frames, and Salesman sequence contains motion occlusions. In order to have better estimations of steering kernel weights, we estimate patchwise (4×4 block) translational motions by the optical flow technique [21], and apply 3-D ISKR to the roughly motion-compensated inputs.

Though we did not discuss temporal upscaling much explicitly in the text of this chapter, the proposed algorithm is capable of this functionality as well in a very straightforward way. Namely, the temporal upscaling is effected by


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

(d) 3-D ISKR ($t = 26.5$) (e) 3-D ISKR ($t = 27$) (f) 3-D ISKR ($t = 27.5$)

FIGURE 3.14: 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 [28], respectively, and (d)-(f) upscaled frames by 3-D ISKR at $t = 26.5$, 27, and 27.5, respectively.

producing a pilot estimate and improving the estimate iteratively just as in the spatial upscaling case illustrated in the block diagrams in Figure 3.8. We note that this temporal upscaling capability, which essentially comes for free in our present framework, was not possible in the NL-means based algorithm [28].

The first example in Figure 3.14 is a real experiment[7] of space-time upscaling with a native Carphone sequence in QCIF format (144×176, 30 frames). Figure 3.14 shows (a) the input frame at time $t = 27$ and (b)-(c) the upscaled frames by Lanczos interpolation and NLM-based method [28], and (d)-(f) the upscaled frames by 3-D ISKR at $t = 26.5$, 27 and 27.5, respectively. The estimated frames in Figure 3.14(d)-(f) shows the application of 3-D ISKR, namely simultaneous space-time upscaling.

The final example shown in Figure 3.15 is also a real example of frame interpolation (temporal upscaling) using Salesman sequence where, although there is no global (camera) motion, the both hands move toward different directions and occlusions can be seen around the tie as shown in the input frames (Figure 3.15(a)). In this example, we estimate intermediate frames at times $t = 6.5$ and $t = 7.5$, and the frames in Figure 3.15(b) are the results by 3-D ISKR. 3-D ISKR successfully generated the intermediate frames without producing any artifacts.

3.4 Conclusion

Traditionally, super-resolution reconstruction of image sequences has relied strongly on the availability of highly accurate motion estimates between the 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, we extended the 2-D steering KR method to an iterative 3-D framework, which works well for both (spatiotemporal) video upscaling and denoising applications. Significantly, we illustrated that the need for explicit subpixel motion estimation can be avoided by the two-tiered approach presented in Section 3.2.4, which yields excellent results in both spatial and temporal upscaling.

Performance analysis of super-resolution algorithm remains an interesting area of work, particularly with the new class of algorithms such as the proposed and NLM-based method [28] which can avoid subpixel motion estimation. Some results already exist which provide such bounds under certain

[7]That is to say, the input to the algorithm was the native resolution video, which was subsequently upscaled in space and time directly. In other words, the input video is *not* simulated by downsampling a higher resolution sequence.

simplifying conditions [29], but these results need to be expanded and studied further.

Reducing the computational complexity of 3-D ISKR is of great importance. Most of the computational load is due to (in order of severity): (i) the computations of steering (covariance) matrices ($\boldsymbol{C}_i$) in (3.22), (ii) the generation of the equivalent kernel coefficients (W_i) in (3.34) from the steering kernel function with higher (i.e., $N = 2$), and (iii) iterations. For (i), to speed up the estimation of $\boldsymbol{C}_i$, instead of application of SVD, which is computationally heavy, we can create a lookup table containing a discrete set of representative

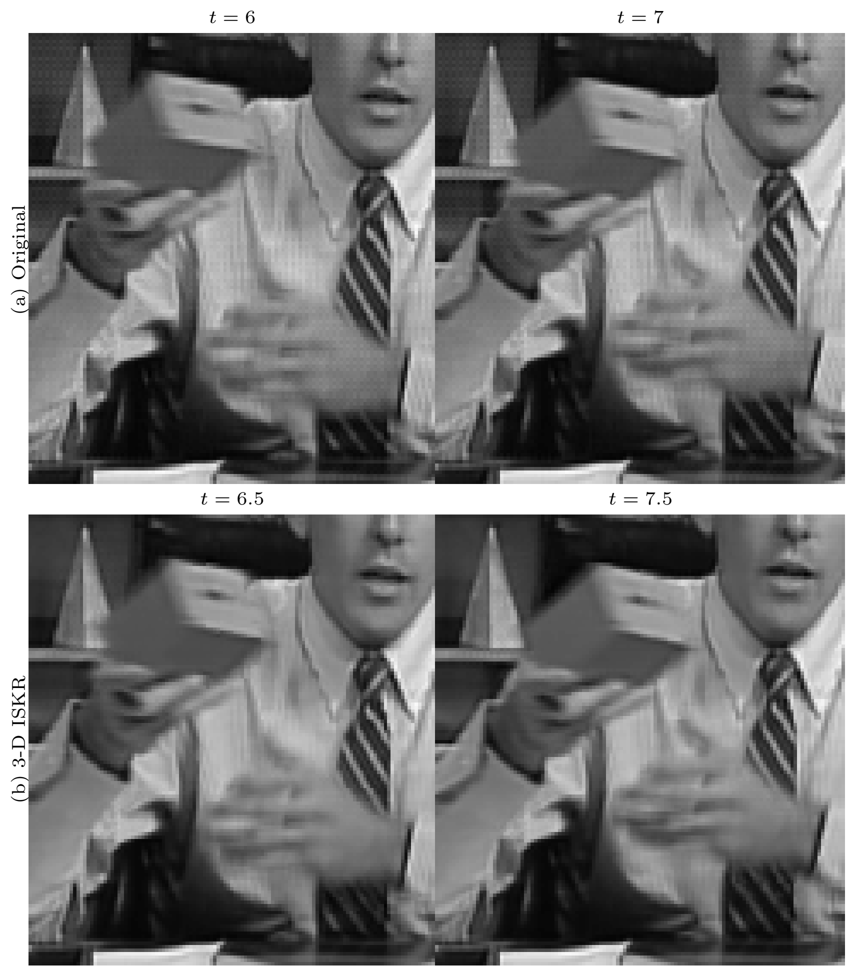


FIGURE 3.15: A Salesman example of frame interpolation: (a) original (input) video frames at time $t = 6$ to 7, (b) intermediate frames estimated by 3-D ISKR at $t = 6.5$ and $t = 7.5$.

steering matrices (using, say, vector quantization), and choose an appropriate matrix from the table given local data. For (ii), computation of the second order ($N = 2$) filter coefficients (W_i) from the steering kernel weights (3.35) may be sped up by using an approximation using the lower order (e.g. zeroth order, $N = 0$) kernels. This idea was originally proposed by Haralick in [13] and may be directly applicable to our case as well. For (iii), we iterate the process of steering kernel regression in order to obtain better estimates of orientations. If the quantization mentioned above gives us fairly reasonable estimates of orientations, we may not need to iterate.

3.5 Appendix

3.5.1 Steering Kernel Parameters

Using the (compact) SVD (3.26) of the local gradient vector $\boldsymbol{J}_i$ (3.23), we can express the naive estimate of steering matrix as:

$$\begin{aligned}
\widehat{\boldsymbol{C}}_i^{\text{naive}} &= \boldsymbol{J}_i^T \boldsymbol{J}_i = \boldsymbol{V}_i \boldsymbol{S}_i^T \boldsymbol{S}_i \boldsymbol{V}_i^T \\
&= \boldsymbol{V}_i \operatorname{diag}\left\{s_1^2, s_2^2, s_3^2\right\} \boldsymbol{V}_i^T \\
&= s_1 s_2 s_3 \boldsymbol{V}_i \operatorname{diag}\left\{\frac{s_1}{s_2 s_3}, \frac{s_2}{s_1 s_3}, \frac{s_3}{s_1 s_2}\right\} \boldsymbol{V}_i^T \\
&= Q\gamma_i \left[\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3\right] \operatorname{diag}\left\{\varrho_1, \varrho_2, \varrho_3\right\} \left[\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3\right]^T \\
&= Q\gamma_i \sum_{q=1}^{3} \varrho_q \boldsymbol{v}_q \boldsymbol{v}_q^T,
\end{aligned} \tag{3.37}$$

where

$$\varrho_1 = \frac{s_1}{s_2 s_3}, \quad \varrho_2 = \frac{s_2}{s_1 s_3}, \quad \varrho_3 = \frac{s_3}{s_1 s_2}, \quad \gamma_i = \frac{s_1 s_2 s_3}{Q}, \tag{3.38}$$

and Q is the number of rows in $\boldsymbol{J}_i$. 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 (3.25) from being zero.

3.5.2 The Choice of the Regression Parameters

The parameters which have critical roles in steering kernel regression are the regression order (N), the global smoothing parameter (h) in (3.22) and (3.27), and the structure sensitivity (α) in (3.25). It is generally known that the parameters N and h control the balance between the variance and bias of the estimator [30]. The larger N and the smaller h, the higher the variance becomes and the loser the bias. In this work, we fix the regression order $N = 2$.

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 (s_1, s_2, s_3) indicates the amount of the 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. In this work, we focus on the case that the given video sequences have a moderate amount of noise and fix $\alpha = 0.1$.

Ideally, although one would like to automatically set these regression parameters using a method such as cross-validation [14, 25], SURE (Stein's unbiased risk estimator) [22] or a no-reference parameter selection [42], this would add significant computational complexity to the already heavy load of the proposed method. So for the examples presented in the chapter, we make use of our extensive earlier experience to note that only certain ranges of values for the said parameters tend to give reasonable results. We fix the values of the parameters within these ranges to yield the best results, as discussed in Section 3.3.

3.5.3 Deblurring

Since we did not include the effect of sensor blur in the data model of the KR framework, deblurring is necessary as a post-processing step to improve the outputs by 3-D ISKR further. Defining the estimated frame at time t as

$$\hat{\underline{\boldsymbol{z}}}(t) = [\cdots, \hat{z}(\boldsymbol{x}_j), \cdots]^T, \tag{3.39}$$

where j is the index of the spatial pixel array and $\boldsymbol{u}(t)$ as the unknown image of interest, we deblur the frame $\underline{\boldsymbol{z}}(t)$ by a regularization approach:

$$\hat{\underline{\boldsymbol{u}}}(t) = \arg\min_{\underline{\boldsymbol{u}}} \left\| \underline{\boldsymbol{u}}(t) - \boldsymbol{G}\, \hat{\underline{\boldsymbol{z}}}(t) \right\|_2^2 + \lambda C_{\mathrm{R}}(\hat{\underline{\boldsymbol{u}}}(t)), \tag{3.40}$$

where $\boldsymbol{G}$ is the blur matrix, $\lambda(\geq 0)$ is the regularization parameter, and $C_{\mathrm{R}}(\,\cdot\,)$ is the regularization term. More specifically, we rely on our earlier work and employ the *bilateral total variation* (BTV) framework [8]:

$$C_{\mathrm{R}}(\underline{\boldsymbol{u}}(t)) = \sum_{v_1=-\nu}^{\nu} \sum_{v_2=-\nu}^{\nu} \eta^{|v_1|+|v_2|} \left\| \underline{\boldsymbol{u}}(t) - \boldsymbol{F}_{x_1}^{v_1} \boldsymbol{F}_{x_2}^{v_2} \underline{\boldsymbol{u}}(t) \right\|_1 \tag{3.41}$$

where η is the smoothing parameter, ν is the window size, and $\boldsymbol{F}_{x_1}^{v_1}$ is the shift matrix that shifts $\underline{\boldsymbol{u}}(t)$ v_1-pixels along x_1-axis.

In the present work, we use the above the BTV regularization framework to deblur the upscaled sequences frame-by-frame, which is admittedly suboptimal. In our work [32], we have introduced a different regularization function called *adaptive kernel total variation* (AKTV). This framework can be extended to derive an algorithm which can simultaneously interpolate and deblur in one integrated step. This promising approach is part of our ongoing work and is outside the scope of this chapter.

Bibliography

[1] M. J. Black and P. Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 63(1):75–104, January 1996.

[2] 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*, 4(2):490–530, 2005.

[3] S. Chaudhuri and S. Chatterjee. Performance analysis of total least squares methods in three-dimensional motion estimation. *IEEE Transactions on Robotics and Automation*, 7(5):707–714, October 1991.

[4] K. Dabov, A. Foi, V. Katkovnic, and K. Egiazarian. Image denoising by sparse 3D transform-domain collaborative filtering. *IEEE Transactions of Image Processing*, 16(8):2080–2095, August 2007.

[5] 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.

[6] 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*, 10(8):1187–1193, August 2001.

[7] Gunnar Farnebäck. *Polynomial Expansion for Orientation and Motion Estimation.* PhD thesis, Linköping University, Sweden, SE-581 83 Linköping, Sweden, 2002. Dissertation No 790, ISBN 91-7373-475-6.

[8] S. Farsiu, D. Robinson, M. Elad, and P. Milanfar. Fast and robust multi-frame super-resolution. *IEEE Transactions on Image Processing*, 13(10):1327–1344, October 2004.

[9] W. T. Freeman, T. R. Jones, and E. C. Pasztor. Example-based super resolution. *IEEE Computer Graphics*, 22(2):56–65, March/April 2002.

[10] H. Fu and J. Barlow. A regularized structured total least squares algorithm for high-resolution image reconstruction. *Linear Algebra and its Applications*, 391:75–98, November 2004.

[11] J. J. Gibson. *The Perception of the Visual World.* Houghton Mifflin, Boston, 1950.

[12] B. K. Gunturk, Y. Altunbasak, and R. M. Mersereau. Multiframe resolution enhancement methods for compressed video. *IEEE Signal Processing Letter*, 9:170–174, June 2002.

[13] R. M. Haralick. Edge and region analysis for digital image data. *Computer Graphic and Image Processing (CGIP)*, 12(1):60–73, January 1980.

[14] W. Hardle and P. Vieu. Kernel regression smoothing of time series. *Journal of Time Series Analysis*, 13:209–232, 1992.

[15] B. K. Horn. *Robot Vision.* MIT Press, Cambridge, 1986.

[16] M. Irani and S. Peleg. Super resolution from image sequence. *Proceedings of 10th International Conference on Pattern Recognition (ICPR)*, 2:115–120, 1990.

[17] 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*, 6696:66960T, 2007.

[18] H. Knutsson. Representing local structure using tensors. *Proceedings of the 6th Scandinavian Conference on Image Analysis*, pages 244–251, 1989.

[19] 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)*, pages 515–523, June 1993.

[20] 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*, 290(1038):169–179, 1980.

[21] B. Lucas and T. Kanade. An iterative image registration technique with an application to sterio vision. *Proceedings of DARPA Image Understanding Workshop*, pages 121–130, 1981.

[22] F. Luisier, T. Blu, and M. Unser. A new sure approach to image denoising: Inter-scale orthonormal wavelet thresholding. *IEEE Transactions on Image Processing*, 16(3):593–606, March 2007.

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

[24] 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*, 12:35–42, 2002.

[25] N. Nguyen, P. Milanfar, and G. H. Golub. Efficient generalized cross-validation with applications to parametric image restoration and resolution enhancement. *IEEE Transactions on Image Processing*, 10(9):1299–1308, September 2001.

[26] K. S. Ni, S. Kumar, N. Vasconcelos, and T. Q. Nguyen. Single image superresolution based on support vector regression. *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2, May 2006.

[27] SC. Park, MK. Park, and MG. Kang. Super-resolution image reconstruction: A technical overview. *IEEE Signal Processing Magazine*, 20(3):21–36, May 2003.

[28] M. Protter, M. Elad, H. Takeda, and P. Milanfar. Generalizing the nonlocal-means to super-resolution reconstruction. *IEEE Transactions on Image Processing*, 16(2):36–51, January 2009.

[29] D. Robinson and P. Milanfar. Statistical performance analysis of super-resolution. *IEEE Transactions on Image Processing*, 15(6):1413–1428, June 2006.

[30] D. Ruppert and M. P. Wand. Multivariate locally weighted least squares regression. *The annals of statistics*, 22(3):1346–1370, September 1994.

[31] H. Takeda, S. Farsiu, and P. Milanfar. Kernel regression for image processing and reconstruction. *IEEE Transactions on Image Processing*, 16(2):349–366, February 2007.

[32] H. Takeda, S. Farsiu, and P. Milanfar. Deblurring using regularized locally-adaptive kernel regression. *IEEE Transactions on Image Processing*, 17(4):550–563, April 2008.

[33] 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.

[34] H. Takeda, P. van Beek, and P. Milanfar. Spatiotemporal video upscaling using motion-assisted steering kernel (MASK) regression (Book chapter). *High-Quality Visual Experience: Creation, Processing and Interactivity of High-Resolution and High-Dimensional Video Signals*, Springer-Verlag, 2010.

[35] J. van de Weijer and R. van den Boomgaard. Least squares and robust estimation of local image structure. *Scale Space. International Conference*, 2695(4):237–254, 2003.

[36] P. Vandewalle, L. Sbaiz, M. Vetterli, and S. Susstrunk. Super-resolution from highly undersampled images. *Proceedings of International Conference on Image Processing (ICIP)*, pages 889–892, September 2005. Genova, Italy.

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

[38] K. Q. Weinberger and G. Tesauro. Metric learning for kernel regression. *Proceedings of the Eleventh International Workshop on Artificial Intelligence and Statistics*, pages 608–615, 2007. (AISTATS-07), Puerto Rico.

[39] 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*, 15(1):201–213, January 2006.

[40] 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.

[41] S. Zhu and K. Ma. A new diamond search algorithm for fast block-matching motion estimation. *IEEE Transactions on Image Processing*, 9(2):287–290, February 2000.

[42] 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.

[43] 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.