Multidimensional directional steerable filters - Theory and application to 3D flow estimation

Dimitrios S. Alexiadis[a], Nikolaos Mitianoudis[a], Tania Stathaki[b]

[a] *Democritus University of Thrace, Department of Electrical & Computer Engineering, University Campus Xanthi- Kimmeria 67100 Xanthi, Greece*
[b] *Imperial College of London, Department. of Electrical & Electronic Engineering, Exhibition road, SW7 2AZ, London, U.K*

Abstract

In this paper, a thorough theoretical analysis on the construction of multi-dimensional directional steerable filters is given. Steerable filters have been constructed for up to three dimensions. We extend the relevant theory to multiple dimensions and construct multi-dimensional steerable filters, as well as quadrature pairs of such filters. Formulating the multi-dimensional motion estimation problem in the spatiotemporal frequency domain, it is shown that motion manifests itself as energy concentration along "motion hyper-planes" in that domain. Subsequently, using the constucted multi-dimensional filters, we formulate the "hyper-donut" mechanism, i.e. a mechanism to efficiently "measure" the "motion energy" on a "motion hyper-plane". On top of that, rigorous mathematical analysis on the use of the constructed filters in the dense flow estimation task is given. Based on the theoretical developments, a steerable filter-based algorithm is formulated, in its simplest possible form, for estimating 3D flow in sequences of volumetric or point-cloud data. Experimental results on simulated and real-world data verify the validity of our arguments and the effectiveness of the proposed method.

Keywords: steerable filters, multi-dimensional signal processing, frequency domain, 3D flow estimation

*Corresponding author, Tel: +30 6944 290233, Fax: +30 25410 79569
Email address: dalexiad@gmail.com; dalexiad@iti.gr (Dimitrios S. Alexiadis)

1. Introduction

Directional filters have been applied in many computer vision and image processing tasks, including edge detection, texture analysis, image compression and 2D motion analysis. In such tasks, similar filters oriented at various directions, are applied to the input 2D or 3D signal under adaptive control and the filters' output is examined as a function of orientation. The most straightforward approach to compute the responses of a directional filter at many different orientations is to apply many versions of the same filter that differ only by a rotation. However, Freeman and Adelson [1, 2] introduced the notion of "steerable" directional filters, to describe a class of filters in which a filter at an arbitrary orientation can be interpolated ("steered") as a linear combination of a few predefined "basic filters". This enables the calculation of responses at arbitrary orientations by interpolation between the basic filter responses, letting one to develop efficient algorithmic solutions.

Since a filter's output depends on the local phase of the input image, the design and use of quadrature pairs of oriented steerable filters (i.e. pairs of filters that are the Hilbert transform of each other, commonly a real-valued and a complex-valued filter) is also essential, allowing adaptive control over phase and orientation [1], as well as local phase-independent measurements of oriented energy [3].

In this work, we provide a thorough theoretical analysis for the construction and use of multi-dimensional steerable directional filters, as well as quadrature pairs of such filters, generalizing the theory into multiple dimensions. Therefore, in a generic context and from a theoretical point-of-view, the paper provides a toolbox for multi-dimensional signal processing and addresses the problem of "structure" analysis in multi-dimensional signals.

From a practical point-of-view, we provide a spatiotemporal framework for motion (flow) analysis in sequences of multi-dimensional signals. The problem is handled directly in multiple dimensions. The idea is based on the fact that motions in a sequence of N-D data manifest as directional "structures" in the $(N+1)$-D space-time domain. This is a generalization of Adelson and Bergen's idea [4] that 2D motion appears as a linear structure in the 3D space-time and can be handled as such. Motivated by the fact that, according to early studies, human motion perception mechanisms can be modeled based on frequency domain considerations, we formulate and study the problem in that domain. It is shown that N-D motion manifests itself as energy concentration along "motion hyper-planes" in the $(N+1)$-dimensional

frequency domain.

In order to provide an efficient solution to the N-D flow problem, we formulate the "Hyper-donut mechanism" for both simple directional filters and quadrature pairs, a generalization of the "Donut mechanism" [5, 3]. This mechanism "samples" the "motion energy" along equally-distributed directions on a "motion hyper-plane", enabling the efficient use of steerable filters in the flow estimation problem. The generalization of relevant theory to multiple dimensions is not straightforward, due to fact that standard 2D or 3D operations (e.g. cross product) do not always have a corresponding operation in multiple dimensions, while direct visualization in multiple dimensions is not possible.

From an experimentation point-of-view, the problem of 3D flow estimation in sequences of 3D data is addressed. More specifically, we work with sequences of pure volumetric data, such as medical MRI data, as well as with sequences of 3D point-clouds to address the problem of 3D scene-flow [6] estimation for scene analysis and understanding. The formulated algorithms are provided in their simplest possible form to show-case the validity of the developed theory. The proposed algorithms are among the few ones that handle 3D flow directly in three dimensions. Its efficiency is demonstrated by extensive experiments.

1.1. Previous relevant work

Steerable filters: Freeman and Adelson in [1, 2] elucidated an elegant theory for steerable filters, constructed 2D and 3D filters and quadrature pairs from derivatives of Gaussians (DOG filters) and demonstrated their use in 2D image analysis tasks. Three-dimensional Gabor steerable filters have been applied to the 2D flow estimation problem in [7]. Three-dimensional separable DOG steerable filters have been also used in [8] for motion analysis in video surveillance tasks. Andersson [9] constructed 3D directional cosine filters of order $L \leq 2$ (wide filters) and presented the corresponding ideas for the construction of quadrature pairs. Simoncelli [5], introducing the "Donut mechanism" for motion estimation, highlighted the importance of narrow directional filters for the computation of multiple motions, without however paying much attention to filters' steerability. In [3], the theory was generalized towards the construction of 3D steerable directional cosine filters and quadrature pairs of arbitrary high order (narrow filters). The "Donut mechanism" was also extended for quadrature pairs and the theory was applied for

2D motion estimation. In the current work, we generalize the relevant theory towards the construction of N-dimensional, arbitrary narrow, directional steerable filters and quadrature pairs, as well as we formulate a "Hyper-donut mechanism" for N-D flow estimation. A small portion of the work on the construction of N-D directional filters was presented in our preliminary paper [10], where the 3D flow task was also handled by a simple algorithm that according to our experimental experience is inferior to the one proposed in this paper.

Three-dimensional flow estimation: Although the problem of 2D flow estimation in image sequences has been extensively studied during the last decades [11, 12, 13, 14, 3], only few recent works deal with 3D motion estimation directly in 4D data (3D+Time). These few approaches are based on extending standard well-known differential 2D approaches [11, 12, 13] or block-based matching. More specifically, in [15, 16] the well-known Horn-Schunck [11] and Lucas-Kanade [12] methods have been extended in three dimensions. They are applied in [16] to experiment with gated MRI datasets and calculate the 3D flow for one synchronized heart beat. The 3D Lucas-Kanade method has been also applied in [17] for motion analysis in T1-weighted MRI volume sequences. The same methods, i.e. the 3D local least squares framework (Lucas-Kanade) and the global regularization framework (Horn-Schunck) have been employed in [18, 19] to compute the 3D velocity field from radial velocity measures of a Doppler radar. Three-dimensional motion estimation was employed in [20] for the 4D medical data compression task. In that work, motion estimation is performed by 3D block (cube) matching. However, it is known that block-matching techniques do not necessarily output the actual flow field and are appropriate mainly for compression tasks. Finally, in [21] a combinational 3D matching (correlation) and differential method is proposed for 3D motion estimation in "Particle Image Velocimetry (PIV)" data. However, experimental results on synthetic data are only presented.

We highlight here that only the above referenced methods are similar to the proposed one, in the sense that they take the same input (i.e. a sequence of volumetric data) and output the same kind of data, while they serve similar tasks (apart from block-matching techniques that serve mainly for compression). Among the available methods, we selected to compare against a differential one and specifically the Lucas-Kanade (LK) method, which is known to outperform the differential Horn-Schunck and region-matching

methods in the 2D case [13].

Finally, in this subsection, we have to mention the "range flow" estimation method [22], which extends standard 2D differential methods for computing the flow in sequences of depth (partial 3D) images, rather than volume data.

Less relevant methods - Scene-flow: In this paper, we also address the problem of motion estimation in sequences of 3D point-clouds. This is similar to the "scene-flow" estimation problem.

The term "scene-flow" was introduced in [6] and refers to the flow that describes the motion at every surface point of a 3D scene. In that paper, the scene-flow is recovered from multi-view RGB image sequences. To extract the scene-flow, the 2D optical flow in each of the input images is estimated using standard 2D optical-flow techniques and the 3D flow is extracted by formulating and solving an over-determined system of equations for each voxel occupied by a surface point. In another work [23], the scene flow is extracted from multi-view image sequences and used to calculate local 3D motion descriptors for human action recognition. The 2D optical-flow is initially estimated in each RGB image and the scene flow is extracted from its 2D projections. Most of the existing estimation methods, similarly to the referenced papers, extract the scene-flow from its 2D flow projections, which is a relatively fast but approximate solution. In addition, the problem of fusing 2D flows from multiple cameras has to be handled, which for example requires visibility (occlusion) information. Although such methods compute initially the flow on the image plane, they require the 3D scene geometry to be first reconstructed, in order to map the 2D flows onto 3D.

One could additionally associate the 3D flow estimation task with the problem of registering sequential 3D captures of non-rigidly deforming objects. Therefore, one might refer here to a class of non-rigid 3D registration methods, such as [24, 25], which solve for correspondences between points on a source and a target scan and estimate a warping field that brings the source scan into alignment with the target one. However, such methods can handle specific kinds of 3D geometries, while they serve the specific task of 3D scan registration, rather than scene understanding tasks.

Compared to the methods referenced in this paragraph, the proposed method is more generic in the sense that it is formulated for volumetric input sequences and therefore it can serve additional tasks, such as flow estimation in medical 3D image sequences.

2. Theoretical developments: Multi-dimensional flow in the frequency domain

2.1. Flow in the frequency domain

Consider a scalar-valued function $f_0(\mathbf{x}_s)$, where $\mathbf{x}_s$ is the spatial coordinates vector. Let also its evolution with time be denoted as $f(\mathbf{x}_s;t)$, such that it equals $f_0(\mathbf{x}_s)$ at time t=0. We consider the general case where $f(\mathbf{x}_s;t)$ is a function of the N-dimensional space-time. For $N=3$, it corresponds to a sequence of images and for $N=4$ to a sequence of volumes.

In the simplest flow model, we assume that the flow in a small spatiotemporal neighborhood is approximated by a single translational velocity vector $\mathbf{v}_o$, namely $f(\mathbf{x}_s;t) = f_0(\mathbf{x}_s - \mathbf{v}_o t)$. Considering the spatial Fourier-Transform (FT) for each time instance, due to the shift property of FT, it holds: $\tilde{f}(\boldsymbol{\omega}_s;t) = F_0(\boldsymbol{\omega}_s)e^{-\mathtt{j}\,\boldsymbol{\omega}_s^{\mathtt{T}}\mathbf{v}_o\,t}$, where $\boldsymbol{\omega}_s$ denotes the (N-1)-D spatial frequency vector, $\mathtt{j}=\sqrt{-1}$ and $F_0(\boldsymbol{\omega}_s)$ is the spatial FT of $f_0(\mathbf{x}_s)$. Taking also the temporal FT, it is straightforward to show that:

$$|F(\boldsymbol{\omega}_s;\omega_t)| = |F_0(\boldsymbol{\omega}_s)|\;\delta(\omega_t + \boldsymbol{\omega}_s^{\mathtt{T}}\mathbf{v}_o), \tag{1}$$

where δ denotes the delta function.

Conclusion #1: The energy in the N-D spatiotemporal frequency domain is concentrated along a hyper-plane through the origin $\omega_t + \boldsymbol{\omega}_s^{\mathtt{T}}\mathbf{v}_o = 0$, or equivalently

$$\textit{Motion hyper-plane:} \qquad \boldsymbol{\omega}^{\mathtt{T}} \cdot [\mathbf{v}_o^{\mathtt{T}}, 1]^{\mathtt{T}} = 0, \tag{2}$$

where $\boldsymbol{\omega} = [\boldsymbol{\omega}_s^{\mathtt{T}}; \omega_t]^{\mathtt{T}}$ is the N-D spatiotemporal frequency vector. Similarly, from now on, $\mathbf{x} = [\mathbf{x}_s^{\mathtt{T}}; t]^{\mathtt{T}}$ is used to denote N-D space-time position.

2.2. Objective function for flow estimation - Rationale behind the use of steerable filters

According to Conclusion #1, the estimation of the unknown velocity vector can be cast as a hyper-plane detection problem in the N-D space. More specifically, based on (1), an objective function of the form $P(\mathbf{v}) = \int_{\boldsymbol{\omega}} |F(\boldsymbol{\omega}_s;\omega_t)|^2\;\delta(\omega_t + \boldsymbol{\omega}_s^{\mathtt{T}}\mathbf{v})\;d\boldsymbol{\omega}$ is maximized at $\mathbf{v}$ equal to the actual velocity.

The rational behind the construction and use of steerable filters is based on the following facts: i) The calculation of the function $P(\mathbf{v})$, as defined previously, requires the calculation of an integral (sum) along a hyper-plane for each candidate velocity; The summation along hyper-planes corresponds

to a generalized Hough transform in the N-D space, which is computationally prohibitive - On the other hand the use of steerable filters can lead to much faster solutions, due to the "steerability" property of filters; ii) Due to deviations from the translational model assumption, in practice the energy in the spatiotemporal FT space is concentrated "near" a motion plane and not "along" the plane. Therefore, it would be beneficial to use a functional that considers the energy around (not along) a motion plane. Using directional steerable filters and formulating the "(Hyper-)dunut" mechanism, the energy inside a (hyper-)donut, aligned with the plane, is measured; iii) Although the filters are constructed in the frequency domain, it is straightforward to consider the filters' responses in the original space-time domain (via the Parseval's equation) and subsequently formulate spatiotemporally local objective functions, appropriate for dense flow-field estimation.

3. Theoretical developments: Multidimensional directional steerable filters and quadrature pairs

3.1. Directional steerable filters

3.1.1. Definition of N-D directional filters

Let $\boldsymbol{\omega} = [\omega_1, \omega_2, \ldots, \omega_N]^{\mathsf{T}}$ denote a N-D frequency vector and $\hat{\boldsymbol{\omega}} = \boldsymbol{\omega}/||\boldsymbol{\omega}||$ the corresponding unit-normalized vector. A N-D directional filter of order L, oriented along the unit vector $\mathbf{d} = [d_1, d_2, \ldots, d_N]^{\mathsf{T}}$ in the N-D frequency domain, is defined by

$$B^L_{\mathbf{d}}(\boldsymbol{\omega}) := G\big(||\boldsymbol{\omega}||\big)\ (\hat{\boldsymbol{\omega}}^{\mathsf{T}} \cdot \mathbf{d})^L = G\big(||\boldsymbol{\omega}||\big)\ ||\boldsymbol{\omega}||^{-L} \left(\sum_{n=1}^{N} \omega_n d_n\right)^L, \qquad (3)$$

where $G\big(||\boldsymbol{\omega}||\big)$ can be any radial function, which will be ignored in the rest of the paper (without loss of generality), since it is independent to direction $\mathbf{d}$. Additionally, for the sake of notational simplicity, we drop the filter's order L, wherever it is implied.

By definition, the introduced directional filters present even or odd symmetry in the frequency domain, for even or odd L, respectively. Consequently, they are real- or imaginary-valued in the original space-time domain, respectively. Additionally, the higher the order L of the filter, the narrower the filter is and consequently its directional selectivity is stronger. In [3], it was explained ([3], sec. 3.2.) and experimentally demonstrated that the use of

narrower filters may lead to more accurate flow estimates in 2D flow, especially near object boundaries. However, this comes at the cost of increased computational effort.

In the next paragraph it is shown that a filter defined as in (3), can be "steered" (interpolated) at an arbitrary orientation from a finite set of linearly-independent directional basic filters.

3.1.2. Basis filters, steerability and interpolation formulas

According to the multinomial expansion theorem [26], (3) can be expanded as follows:

$$B_{\mathbf{d}}(\boldsymbol{\omega}) = ||\boldsymbol{\omega}||^{-L} \sum_{\mathbf{p}} \left(C(p_1, p_2, \ldots, p_N; L) \prod_{n=1}^{N} d_n^{p_n} \prod_{n=1}^{N} \omega_n^{p_n} \right), \tag{4}$$

where $\mathbf{p} = [p_1, p_2, \ldots, p_N]^{\mathtt{T}}$ contains non-negative integers that sum-up to L. In other words, the summation runs for all combination of integers $p_1, p_2 \ldots, p_N \geq 0$ that sum up to L. The number of monomial terms equals

$$I_0(N; L) := \begin{pmatrix} L+N-1 \\ N-1 \end{pmatrix} = \frac{(L+N-1)!}{(N-1)!L!} = \frac{\prod_{n=1}^{N-1}(L+n)}{(N-1)!}, \tag{5}$$

while the expansion coefficients are given from: $C(p_1, p_2, \ldots, p_N; L) := \frac{L!}{p_1! p_2! \ldots p_N!}$. In order to proceed, for notational simplicity, we define the vector:

$$\mathbf{c}(\boldsymbol{\omega}) := ||\boldsymbol{\omega}||^{-L} \quad [\quad \underset{\substack{p_1, .., p_N \geq 0 \\ p_1 + .. + p_N = L}}{} \quad C(p_1, .., p_N; L) \prod_{n=1}^{N} \omega_n^{p_n} \quad]^{\mathtt{T}}. \tag{6}$$

Example: For $N = 4$ and $L = 2$, we have the vector of length $I_0(4; 2) = 10$

$$\mathbf{c}(\boldsymbol{\omega}) := ||\boldsymbol{\omega}||^{-2} [\omega_1^2, 2\omega_1\omega_2, 2\omega_1\omega_3, 2\omega_1\omega_4, \omega_2^2, 2\omega_2\omega_3, 2\omega_2\omega_4, \omega_3^2, 2\omega_3\omega_4, \omega_4^2]^{\mathtt{T}}.$$

Similarly, we define the vector

$$\mathbf{v}(\mathbf{d}) := \quad [\quad \underset{\substack{p_1, .., p_N \geq 0 \\ p_1 + .. + p_N = L}}{} \quad \prod_{n=1}^{N} d_n^{p_n} \quad]^{\mathtt{T}}. \tag{7}$$

Using the above definitions and vector notation, (4) is rewritten as

$$B_{\mathbf{d}}(\boldsymbol{\omega}) = \mathbf{v}(\mathbf{d})^{\mathtt{T}} \cdot \mathbf{c}(\boldsymbol{\omega}). \tag{8}$$

Basis filters: Consider $I \geq I_0(N;L)$ basis filters $B_{\mathbf{d}_i}(\boldsymbol{\omega})$, at the basic orientations $\mathbf{d}_i, i = 1, 2, \ldots, I$. Denote as $\mathbf{B}(\boldsymbol{\omega}) := [B_{\mathbf{d}_1}(\boldsymbol{\omega}), B_{\mathbf{d}_2}(\boldsymbol{\omega}), ..., B_{\mathbf{d}_I}(\boldsymbol{\omega})]^{\mathtt{T}}$ the basis filters' vector. Let also

$$\mathtt{U} := [\mathbf{v}(\mathbf{d}_1)^{\mathtt{T}}, \mathbf{v}(\mathbf{d}_2)^{\mathtt{T}}, \ldots, \mathbf{v}(\mathbf{d}_I)^{\mathtt{T}}]^{\mathtt{T}} \tag{9}$$

be a matrix of size $I \times I_0(N;L)$. Then, using (8) and the above definitions, the set of basis filters is $\mathbf{B}(\boldsymbol{\omega}) = \mathtt{U} \cdot \mathbf{c}(\boldsymbol{\omega})$. Solving for $\mathbf{c}(\boldsymbol{\omega})$, we get:

$$\mathbf{c}(\boldsymbol{\omega}) = \mathtt{U}^{-1} \cdot \mathbf{B}(\boldsymbol{\omega}), \tag{10}$$

where $\mathtt{U}^{-1}$ is the (pseudo-)inverse of $\mathtt{U}$.

Interpolation formula: Using (10), (8) is written as:

$$B_{\mathbf{d}}(\boldsymbol{\omega}) = \mathbf{v}(\mathbf{d})^{\mathtt{T}} \cdot \mathtt{U}^{-1} \cdot \mathbf{B}(\boldsymbol{\omega}) = \mathbf{t}(\mathbf{d}) \cdot \mathbf{B}(\boldsymbol{\omega}) = \sum_{i=1}^{I} t_i(\mathbf{d}) B_{\mathbf{d}_i}(\boldsymbol{\omega}), \tag{11}$$

where $\mathbf{t}(\mathbf{d}) := \mathbf{v}(\mathbf{d})^{\mathtt{T}} \cdot \mathtt{U}^{-1}$ is the interpolation vector of length I. The rank of matrix $\mathtt{U}$ should be at least $I_0(N;L)$ so that $\mathtt{U}$ has a (pseudo-)inverse. This means that in order to have a complete filter basis, it should contain at least $I_0(N;L)$ filters in indepedent directions.

Conclusion #2: A N-D directional filter of order L, oriented at an arbitrary orientation $\mathbf{d}$, defined as (3), can be interpolated from $I \geq I_0(N;L)$ basic directional filters $B_{\mathbf{d}_i}(\boldsymbol{\omega})$, using the interpolation formula (11).

From (5) it is evident that the number of basis filters increases with filter's order L and dimensionality N. For the 4D case ($N = 4$), the minimum number of basic filters $I_0(L)$ is given in the second column of Table 2.

On the selection of basic filters directions: Throughout the strict theoretical analysis of previous paragraph, no constraints have been imposed on the basic unit directions $\mathbf{d}_i$. Indeed, Conclusion #2 is theoretically valid for any selection of the basic filters directions. However, intuitively, distributing the basis filters directions uniformly on the unit hyper-sphere sounds reasonable, e.g it leads to a rotation-invariant system. A relevant analysis for the 3D case is given in [3], justifying a uniform selection of basic directions on the unit sphere. Generally, in practical applications one has to take into account the machine arithmetic precision and quantization noise. As a simple example, finite arithmetic precision may lead to singularities in the calculation of the inverse matrix $\mathtt{U}^{-1}$, if some rows of $\mathtt{U}$ (directions $\mathbf{d}_i$) are close to linearly dependent.

According to the above, we select (approximately) uniformly the basic filters directions on the unit (hyper-)sphere. More specifically, since the problem of distributing a number of points "uniformly" on a (hyper-) sphere can be formulated in various ways, we use the solution to the "hard-spheres problem" (best packing on the sphere) [27], which maximizes the smallest distance among the points.

3.1.3. Filter responses and their "steerability"

As in subsection 2.1, we let $f(\mathbf{x})$ denote a N-D spatiotemporal function and $F(\boldsymbol{\omega})$ the corresponding spatiotemporal FT. Let also $b_{\mathbf{d}}(\mathbf{x})$ denote a directional filter in the original space-time domain, namely the inverse FT of $B_{\mathbf{d}}(\boldsymbol{\omega})$. The filter's response in the frequency domain is denoted as $Y_{\mathbf{d}}(\boldsymbol{\omega}) = B_{\mathbf{d}}(\boldsymbol{\omega})F(\boldsymbol{\omega})$, while in the original space-time domain it is $y_{\mathbf{d}}(\mathbf{x}) = b_{\mathbf{d}}(\mathbf{x}) * f(\mathbf{x})$, where $(*)$ stands for convolution.

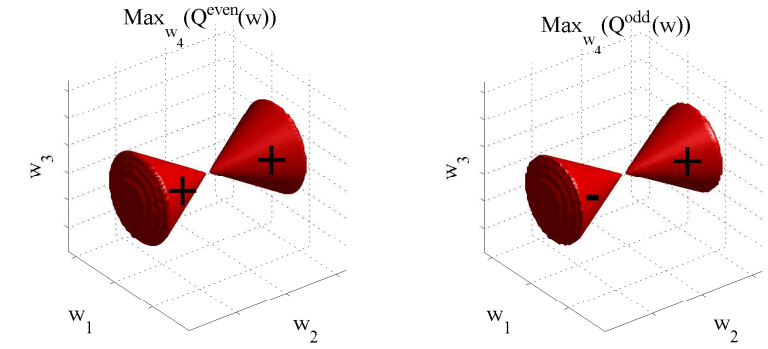
Due to the linearity of FT (or convolution in the original space-time domain), it is straightforward to show that the interpolation scheme of (11) holds also for the filter responses, i.e. the response of a directional filter at an arbitrary orientation can be interpolated from the responses of the basic directional filters, namely $Y_{\mathbf{d}}(\boldsymbol{\omega}) = \sum_{i=1}^{I} t_i(\mathbf{d}) Y_{\mathbf{d}_i}(\boldsymbol{\omega})$. Hereinafter, whenever we refer to the steerability of the filters, the same arguments hold for the responses (and vice-versa).

3.2. Quadrature N-D steerable directional filters

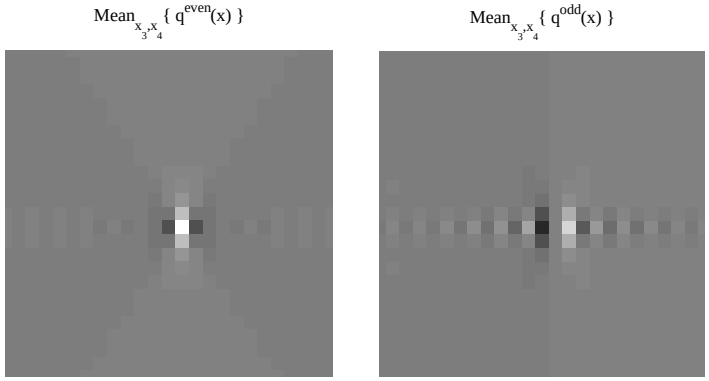
The constructed directional filters are even- or odd-symmetric in the frequency domain, for even or odd order L, respectively. In the 2D case ($N = 2$), this means that filters are sensitive (have maximum response) at positions with either line-like features or step edges, respectively, as with the real (cosine) and imaginary (sine) part of 2D Gabor filters [1]. In the 3D case ($N = 3$), even or odd filters are sensitive to plane-like features or planar step edges, respectively, and so on. In other words, the response of the filters are local phase-dependent. In order to enable a local phase-independent analysis, quadrature filter pairs are constructed, i.e. pairs of even-odd filters that differ in phase by $\pi/2$ along the filters direction [28, ch.13]:

$$Q_{\mathbf{d}}(\boldsymbol{\omega}) = Q_{\mathbf{d}}^{\text{even}}(\boldsymbol{\omega}) + Q_{\mathbf{d}}^{\text{odd}}(\boldsymbol{\omega}) = \begin{cases} 2Q_{\mathbf{d}}^{\text{even}}(\boldsymbol{\omega}) & \text{if } \boldsymbol{\omega} \cdot \mathbf{d} > 0 \\ 0 & \text{if } \boldsymbol{\omega} \cdot \mathbf{d} \leq 0. \end{cases} \tag{12}$$

In the next subsection, it is shown that steerable quadrature filters of order M can be constructed as a linear combination of steerable filters of



(a) $Q_{\mathbf{d}}^{\text{even}}(\boldsymbol{\omega})$ and $Q_{\mathbf{d}}^{\text{odd}}(\boldsymbol{\omega})$



(b) $q_{\mathbf{d}}^{\text{even}}(\mathbf{x})$ and $q_{\mathbf{d}}^{\text{odd}}(\mathbf{x})$

Figure 1: An example of 4D quadrature pair for $M = 4$, oriented along $\mathbf{d} = [0, 1, 0, 0]^{\mathrm{T}}$. (a) The pair in the frequency domain. An isosurface of the max-projection onto the 4th dimension is shown; (b) The pair in the original space-time domain (a zoom around filter's center is drawn). The mean-projection onto the 3rd and 4th dimensions is shown.

orders $0 \leq L \leq M$:

$$Q_{\mathbf{d}}(\boldsymbol{\omega}) = \sum_{L=0}^{M} \alpha_M[L] B_{\mathbf{d}}^{L}(\boldsymbol{\omega}). \tag{13}$$

A thorough theoretical analysis on the calculation of the coefficients $\alpha_M[L]$ is given in the next subsection. In case the reader wants to skip the theoretical details, the values of the coefficients for $N = 4$ are given in Table 1.

An example of 4D steerable quadrature pairs for $M = 4$ is depicted in Figure 1. One can observe the directional selectivity of the filters and the even/odd symmetry.

Table 1: Coefficients $\mathbf{a}_M$ for the construction of 4D quadrature filters

Order M	$\alpha_M[0]$	$\alpha_M[1]$	$\alpha_M[2]$	$\alpha_M[3]$	$\alpha_M[4]$	$\alpha_M[5]$	$\alpha_M[6]$
2	0.1187	0.6920	0.7121				
3	0.0265	0.3093	0.7843	0.5371			
4	0.0055	0.1076	0.4872	0.7715	0.3946		
5	0.0011	0.0323	0.2269	0.6123	0.7006	0.2858	
6	0.0002	0.0088	0.0883	0.3562	0.6765	0.6046	0.2050

3.2.1. Calculation of coefficients α_M

To find the coefficients $\alpha_M[L]$, without loss of generality, we restrict the analysis for filters oriented along the N-th (last) dimension, i.e along the vector $\mathbf{i}_N = [0, \ldots, 0, 1]^\intercal$. Then, by filters' definition and making use of hyper-spherical coordinates [29, ch. 22] (see Appendix A.1), the filter is $B^L_{\mathbf{i}_N}(\boldsymbol{\omega}) = \hat{\boldsymbol{\omega}}^\intercal \cdot \mathbf{i}_N = \cos^L(\phi_{N-2})$, where ϕ_{N-2} represents the last hyper-spherical angle. Therefore, the quadrature filter to be constructed is of the form:

$$Q_{\mathbf{i}_N}(\boldsymbol{\omega}) = Q(\phi_{N-2}) = \sum_{L=0}^{M} \alpha_M[L] \cos^L(\phi_{N-2}). \tag{14}$$

For notational simplicity in this subsection, we notate ϕ_{N-2} simply as ϕ. According to the surface-integrals theory [29, ch. 22] (see Appendix A.1), the total energy of the filter $Q_{\mathbf{i}_N}(\phi)$ is

$$E_0 = \sigma(S^{N-2}) \int_0^{\pi} \left(\sum_{L=0}^{M} \alpha_M[L] \cos^L \phi \right)^2 \sin^{N-2} \phi \; d\phi, \tag{15}$$

where $\sigma(S^{N-2})$ is the surface of the S^{N-2} sphere. The energy in the rear Fourier half-space, denoted as E_1, is given by the same formula but with integration in the interval $[\pi/2, \pi]$. From the definition (12) of the quadrature filters, the energy E_1 has to be zero. Therefore, we search for the coefficients $\alpha_M[L]$ that minimize E_1, given that the total filter's energy E_0 is equal to unity.

From equation (15), after some manipulations and interchanging summa-

tions and integration, we have:

$$E_0 = \sum_{k=0}^{M} \alpha_M[k] \sum_{l=0}^{M} \alpha_M[l]\ R_0(k,l), \text{ where}$$
$$R_0(k,l) = \sigma(S^{N-2}) \int_0^{\pi} \cos^{k+l}\phi\ \sin^{N-2}\phi\ d\phi. \tag{16}$$

Although numerical integration methods can be used to calculate the integrals $R_0(k,l)$, we derived the analytical closed-form solution for $N = 4$, which can be found in Appendix A.2.

One can use vector-matrix notation, to write (16) as follows:

$$E_0 = \mathbf{a}_M^{\mathtt{T}}\ \mathtt{R}_0\ \mathbf{a}_M, \tag{17}$$

where $\mathtt{R}_0$ is the square and symmetric $(M+1)\times(M+1)$ matrix with elements $R_0(k,l)$ and $\mathbf{a}_M = \left[\alpha_M[0], \alpha_M[1], \ldots, \alpha_M[M]\right]^{\mathtt{T}}$. Following exactly the same analysis, the energy E_1 is written $E_1 = \mathbf{a}_M^{\mathtt{T}}\ \mathtt{R}_1\ \mathbf{a}_M$, where $\mathtt{R}_1$ is defined accordingly, similarly to $\mathtt{R}_0$.

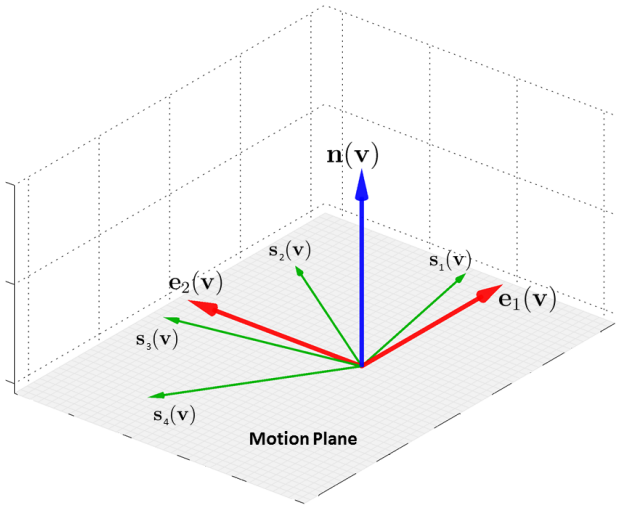
We factorize the symmetric matrix $\mathtt{R}_0$ using its LDL decomposition, i.e. $\mathtt{R}_0 = \mathtt{L}^{\mathtt{T}}\ \mathtt{D}\ \mathtt{L}$ and (17) is written $E_0 = \mathbf{a}_M^{\mathtt{T}}\ \mathtt{L}^{\mathtt{T}}\ \mathtt{D}\ \mathtt{L}\ \mathbf{a}_M$, where D is a diagonal matrix. The restriction $E_0 = 1$ holds if $\mathbf{a}_M$ is selected equal to $\mathbf{a}_M = \mathtt{W}\,\mathbf{i}$, where $\mathtt{W} = \mathtt{L}^{-1}\ \mathtt{D}^{-1/2}$ and $\mathbf{i}$ is any unit vector. Substituting $\mathbf{a}_M$ into E_1 yields $E_1 = \mathbf{i}^{\mathtt{T}}\ \mathtt{W}^{\mathtt{T}}\ \mathtt{R}_1\ \mathtt{W}\,\mathbf{i}$. This is minimized if $\mathbf{i}$ is selected equal to the eigenvector of $\mathtt{W}^{\mathtt{T}}\ \mathtt{R}_1\ \mathtt{W}$ with the minimum eigenvalue. Let this eigenvector be denoted as $\mathbf{i}_m$. Concluding, the desired vector $\mathbf{a}_M$ is given from $\mathbf{a}_M = \mathtt{W}\,\mathbf{i}_m$.

4. Theoretical developments towards motion estimation

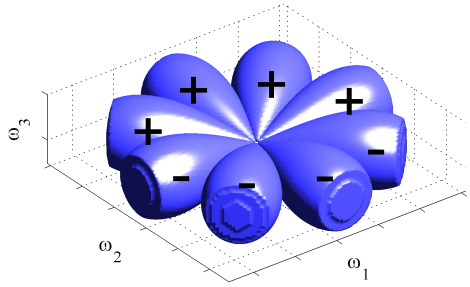
Recalling from subsection 2.1, equation (2), the N-D spatiotemporal power spectrum of a function that translates with velocity $\mathbf{v}_0$ lies on a "motion (hyper-)plane", which is perpendicular to the N-D vector $[\mathbf{v}_0^{\mathtt{T}}, 1]^{\mathtt{T}}$, i.e. it is perpendicular to the unit vector:

$$\textit{Normal to motion (hyper-)plane:} \qquad \mathbf{n}(\mathbf{v}_0) = \frac{[\mathbf{v}_0^{\mathtt{T}}, 1]^{\mathtt{T}}}{\sqrt{|\mathbf{v}_0|^2 + 1}}. \tag{18}$$

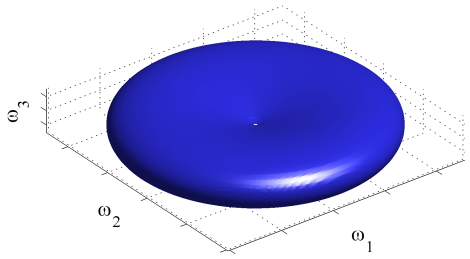
Therefore, to enable motion estimation, our aim is to introduce an objective function $P(\mathbf{v})$ that effectively "measures" the energy along candidate "motion (hyper-)planes" and consequently is maximized at $\mathbf{v}$ equal to the actual



(a) $\mathbf{s}_k(\mathbf{v}), k = 1, \ldots, K$



(b) $B_{\mathbf{s}_k(\mathbf{v})}(\boldsymbol{\omega}), k = 1, \ldots K$



(c) $\sum_{k=1}^{K} \left(B_{\mathbf{s}_k(\mathbf{v})}(\boldsymbol{\omega}) \right)^2$

Figure 2: Illustration of the "(Hyper-)Donut" mechanism in the 3D case ($N = 3$), for $\mathbf{v} = [0, 0]^\mathsf{T}$. The directions of the 3D filters, as well as the "(hyper-)donut", are perpendicular to the vector $\mathbf{n}(\mathbf{v}) = [0, 0, 1]^\mathsf{T}$. We use filters of order $L = 3$. In the 3D case, $K(L) = L+1 = 4$ [3]. Details: (a) Two vectors, $\mathbf{e}_1$ and $\mathbf{e}_2$ span the plane perpendicular to $\mathbf{n}(\mathbf{v})$. From these vectors, the equally distributed vectors $\mathbf{s}_k(\mathbf{v})$ are obtained; (b) The filters $B_{\mathbf{s}_k(\mathbf{v})}(\boldsymbol{\omega})$, as shown, are odd-symmetric in the frequency domain, since L is odd; (c) The sum of the squared filters forms the "Donut". In (b), (c) the -3dB isosurface of is shown.

velocity $\mathbf{v}_0$. Refer also to subsection 2.2 for a discussion on this. The introduced objective function will be referred to as "Max-Steering distribution".

In order to introduce an efficiently calculable objective function, we exploit the constructed filters and their steerability property. Towards this end, we introduce the "Hyper-Donut" mechanism, a generalization of the "Donut" mechanism [5, 3]. This mechanism, exploiting the directional filters, aims at "sampling" the energy along equally distributed directions on a motion plane and subsequently calculating a measure of the energy around this plane (inside a "donut"). The basic idea for the 3D case is illustrated in Fig. 2. Since illustrations are difficult in $N > 3$ dimensions, for easier understanding of the following concepts, the reader is encouraged to study the theory initially for $N = 3$.

4.1. The "Hyper-Donut" Mechanism and "Max-Steering" distribution

Similarly to the 3D case, for a candidate velocity $\mathbf{v}$ in the N-D case, one has to find K unit direction vectors $\mathbf{s}_k(\mathbf{v})$, $k = 1, \ldots, K$, which are equally distributed on the corresponding "motion (hyper-)plane". Towards this, one has to initially find a set of unit vectors $\{\mathbf{e}_n(\mathbf{v})\}_{n=1,\ldots,N-1}$, which spans the subspace that is perpendicular to $\mathbf{n}(\mathbf{v})$, namely it spans the candidate "motion (hyper-)plane". For example, with $N = 3$, two vectors are needed (span the "motion plane"), as shown in Fig. 2(a), while in the 4D case, three vectors are needed to span the "motion hyper-plane".

Let the standard basis of $\mathbb{R}^N$ be $\{\mathbf{i}_1, \mathbf{i}_2, \ldots, \mathbf{i}_N\}$. The needed set of vectors is recursively calculated from

$$\mathbf{e}_n(\mathbf{v}) = \times\Big(\mathbf{n}(\mathbf{v}), \mathbf{e}_1(\mathbf{v}), \ldots, \mathbf{e}_{n-1}(\mathbf{v}), \mathbf{i}_n, \ldots, \mathbf{i}_{N-2}\Big), \tag{19}$$

where the extended cross-product operator $\times(\circ, \circ, \ldots, \circ)$ is defined in N dimensions as the operator that takes $N-1$ vectors and produces a new one which is perpendicular to them [30, ch.4]. In each recursion, the calculated vector $\mathbf{e}_n(\mathbf{v})$ is perpendicular to all the previous $n-1$ vectors and $\mathbf{n}(\mathbf{v})$. For details on the calculation of the extended cross-product $\times(\circ, \circ, \ldots, \circ)$, refer to [30, ch.4] or section A.3 of the supplementary-material document.

Since a set $\{\mathbf{e}_n(\mathbf{v})\}_{n=1,\ldots,N-1}$ is now available, namely a set of vectors that spans the (hyper-)plane subspace, one can get K unit vectors $\mathbf{s}_k(\mathbf{v})$ that are equally distributed on the "motion (hyper-)plane", as follows:

$$\mathbf{s}_k(\mathbf{v}) = \big[\mathbf{e}_1(\mathbf{v}), \mathbf{e}_2(\mathbf{v}), \ldots, \mathbf{e}_{N-1}(\mathbf{v})\big] \cdot \mathbf{q}_k, \quad k = 1, 2, \ldots, K, \tag{20}$$

Table 2: In the 4D case ($N = 4$), the number of basic filters I_0(L) (second column), the number of filters $K(L)$ needed to form a "hyper-donut" (third column) and error measures for the validity of equation (B.1) (5th and 6th column).

Order L	$I_0(L)$	$K(L)$	const(L)	$\frac{\max\{\lvert n(\hat{\boldsymbol{\omega}};L)\rvert\}}{\text{const}(L)}$	$20\log\left(\frac{\text{std}(n(\hat{\boldsymbol{\omega}};L))}{\text{const}(L)}\right)$
1	4	6	2	0	$-\infty$ dB
2	10	12	2.4	0	$-\infty$ dB
3	20	32	4.57	$8 \cdot 10^{-3}$	-51.5 dB
4	35	32	3.56	$17 \cdot 10^{-3}$	-44.7 dB

where $\mathbf{q}_k$ is the k-th unit direction vector from a collection of vectors that are "uniformly" distributed on the (N-1)-dimensional hyper-sphere. For instance, with $N = 3$ and $N = 4$, the vectors $\mathbf{q}_k$ are uniformly distributed on the unit-circle and the unit-sphere, respectively.

4.1.1. The "Hyper-donut" mechanism

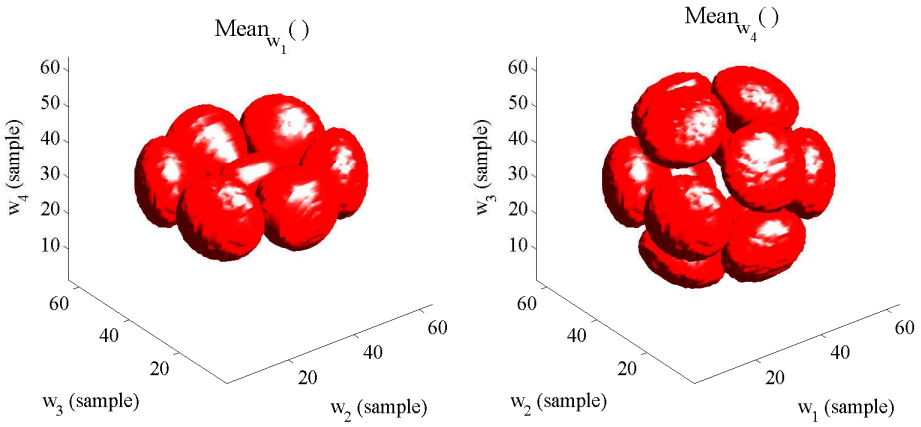
We would like to show at this point, that with an appropriate selection of the number $K(L)$ of vectors $\mathbf{q}_k$, it holds:

$$\textit{On motion hyper-plane } (\forall \boldsymbol{\omega} \perp \mathbf{n}(\mathbf{v})):$$
$$\sum_{k=1}^{K(L)} \left(B^L_{\mathbf{s}_k(\mathbf{v})}(\boldsymbol{\omega})\right)^2 = \sum_{k=1}^{K(L)} \left(\hat{\boldsymbol{\omega}}^{\mathrm{T}} \cdot \mathbf{s}_k(\mathbf{v})\right)^{2L} = \text{const}(L), \tag{21}$$

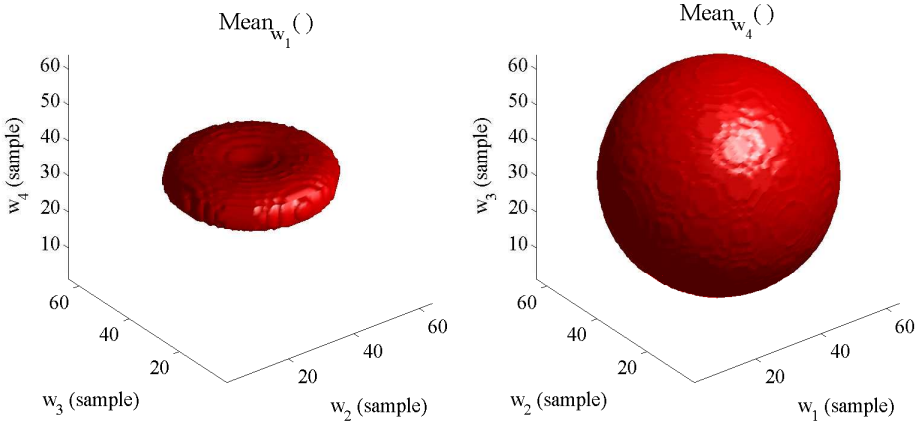
where const(L) is a constant that depends only on the filters' order L, namely the outcome of the summation is constant for each $\boldsymbol{\omega}$ lying on the candidate "motion plane". This would mean that $K(L)$ directional filters, oriented along the directions $\mathbf{s}_k(\mathbf{v})$, $k = 1, \ldots, K(L)$, can effectively "sample" the energy along the "motion hyper-plane" and the energy around the motion plane can be measured by the summation in (21).

In correspondence with the "Donut" mechanism (N=3), given in Fig. 2, the "Hyper-donut" mechanism in the 4D case (N=4) is pictorially explained in Figures 3 and 4. As shown, the filters and the corresponding "Hyper-donut" present "selectivity" to the candidate motion "hyper-plane".

Equation (21) above has been mathematically proved for $N = 3$ in [3], with $K(L) = N + 1$. For the purposes of this work, we have verified that (21) holds also for $N = 4$. Indeed, in the 4D case ($N = 4$), using the values $K(L)$ in the third column of Table 2, it was verified that the equality in (21)

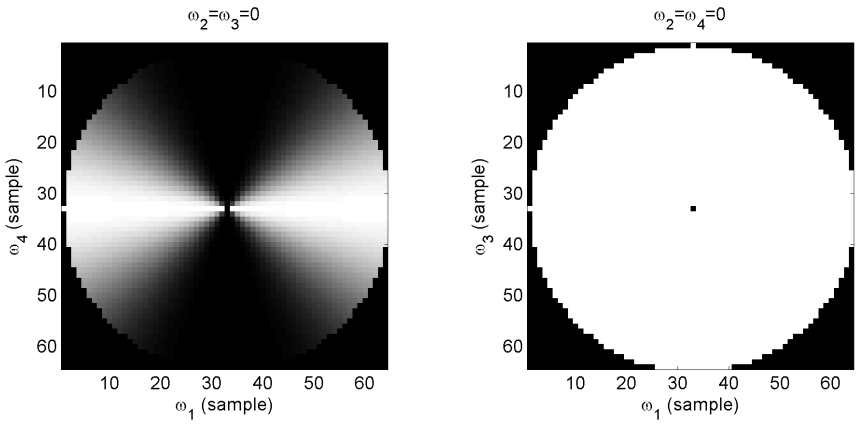


(a) $B_{\mathbf{s}_k(\mathbf{v})}(\boldsymbol{\omega}), k = 1, \ldots, K$. Mean-projections along ω_1 (left) and ω_4 (right).



(b) $\sum_{k=1}^{K} \left(B_{\mathbf{s}_k(\mathbf{v})}(\boldsymbol{\omega}) \right)^2$. Mean-projections along ω_1 (left) and ω_4 (right).

Figure 3: Illustration of the "Hyper-Donut" mechanism in the 4D case ($N = 4$), for $\mathbf{v} = [0, 0, 0]^{\mathsf{T}}$. The motion "hyper-plane", the directions of the 4D filters, as well as the "hyper-donut", are perpendicular to the vector $\mathbf{n}(\mathbf{v}) = [0, 0, 0, 1]^{\mathsf{T}}$. We use filters of order $L = 2$. In this case, $K(L) = 12$. In order to depict the 4D data, mean-projections are depicted. Details: (a) Two mean-projections of the the K directional filters that lie on the "motion hyper-plane"; (b) Two mean-projections of the "hyper-donut", formed by the filters.



(a) $\sum_{k=1}^{K}\left(B_{\mathbf{s}_k(\mathbf{v})}(\boldsymbol{\omega})\right)^2$. Slice $\omega_2{=}\omega_3{=}0$.

(b) $\sum_{k=1}^{K}\left(B_{\mathbf{s}_k(\mathbf{v})}(\boldsymbol{\omega})\right)^2$. Slice $\omega_2{=}\omega_4{=}0$.

Figure 4: Illustration of the "Hyper-Donut" mechanism (continuing from Fig. 3). Here, two slices of of the "hyper-donut" are given. The "hyper-donut" is "perpendicular" to $\mathbf{n}(\mathbf{v}) = [0, 0, 0, 1]^{\mathsf{T}}$ and selective to velocity $\mathbf{v} = [0, 0, 0]^{\mathsf{T}}$.

holds exactly for $L = 1$ and $L = 2$, while it holds approximately for $L = 3$ and $L = 4$ up to very small error $n(\hat{\boldsymbol{\omega}}; L)$. The values of the relative max error and the error-to-signal ratio are given in the 5^{th} and 6^{th} column of the table.

The relevant analysis with respect to the demonstration of equation (21) are given in Appendix B.1.

4.1.2. The Max-Steering distribution

Given that equation (21) holds and based on the "Hyper-donut" concept, one can define the "Max-Steering" objective function

$$P(\mathbf{v}) := \sum_{\boldsymbol{\omega}} \sum_{k=1}^{K} \left|Y_{\mathbf{s}_k(\mathbf{v})}(\boldsymbol{\omega})\right|^2 = \sum_{\boldsymbol{\omega}} \sum_{k=1}^{K} \left|B_{\mathbf{s}_k(\mathbf{v})}(\boldsymbol{\omega}) \cdot F(\boldsymbol{\omega})\right|^2, \qquad (22)$$

which is theoretically maximized at $\mathbf{v}$ equal to the actual velocity. This objective function is in correspondence with the energy function defined in subsection 2.2. More specifically, it is a measure of the spatiotemporal energy inside the "hyper-donut" aligned with the "motion hyper-plane" $\boldsymbol{\omega}^{\mathsf{T}} \cdot [\mathbf{v}^{\mathsf{T}}, 1] = 0$.

Using Parseval's equation, the "Max-Steering" distribution can be rewrit-

ten as

$$P(\mathbf{v}) := \sum_{\mathbf{x}} \sum_{k=1}^{K} \left| y_{\mathbf{s}_k(\mathbf{v})}(\mathbf{x}) \right|^2 = \sum_{\mathbf{x}} \sum_{k=1}^{K} \left| (b_{\mathbf{s}_k(\mathbf{v})} * f)(\mathbf{x}) \right|^2, \tag{23}$$

where $(*)$ denotes convolution.

Summary: To summarize, the value of the "Max-Steering" distribution for a candidate velocity $\mathbf{v}$ can be calculated as follows: i) Using equations (19)-(20), calculate K vectors $\mathbf{s}_k(\mathbf{v})$, that are "equally" distributed on the corresponding "motion hyper-plane"; ii) Calculate the responses of the K directional filters $b_{\mathbf{s}_k(\mathbf{v})}(\mathbf{x})$, oriented at directions $\mathbf{s}_k(\mathbf{v})$; iii) Add the squared responses of the K filters, to measure the spatiotemporal energy around the candidate "motion hyper-plane".

According to the above, for each candidate velocity $\mathbf{v}$, one has to find the responses of K directional filters. Although these responses can be interpolated from the I basic responses (as described in paragraphs 3.1.2-3.1.3), the computational effort increases significantly as the value of K goes high. Fortunately, as shown in the next paragraph, an interpolation formula holds also for the Max-Steering distribution.

4.1.3. Steerability of Max-Steering distribution and interpolation formula

It is shown here that $P(\mathbf{v})$ can be interpolated at any velocity $\mathbf{v}$, given a set of

$$J(L) = I_0(L) \cdot \big(I_0(L) + 1\big)/2 \tag{24}$$

fixed quadratic terms of the form $z_{i,j} := \sum_{\mathbf{x}} y_{\mathbf{d}_i}(\mathbf{x}) \cdot y_{\mathbf{d}_j}(\mathbf{x})$, where $y_{\mathbf{d}_i}(\mathbf{x})$ stands for the i-th basis filter response, as defined in paragraph 3.1.3.

We take into account the fact that the filter responses in the original space-time domain are either pure real or pure imaginary, for even or odd filters, respectively. Additionally, the interpolation formula (in the original space-time domain) is used. Then, manipulations on (23) give:

$$\begin{aligned} P(\mathbf{v}) = \left| \sum_{k=1}^{K} \sum_{\mathbf{x}} y^2_{\mathbf{s}_k(\mathbf{v})}(\mathbf{x}) \right| &= \left| \sum_{k=1}^{K} \sum_{\mathbf{x}} \Big[\sum_{i=0}^{I} t_i\big(\mathbf{s}_k(\mathbf{v})\big) \cdot y_{\mathbf{d}_i}(\mathbf{x}) \Big]^2 \right| \\ &= \left| \sum_{k=1}^{K} \sum_{i=1}^{I} \sum_{j=1}^{I} T_{i,j}\big(\mathbf{s}_k(\mathbf{v})\big) \cdot z_{i,j} \right|, \qquad \square \end{aligned} \tag{25}$$

where $T_{i,j}(\mathbf{d}) := t_i(\mathbf{d}) \cdot t_j(\mathbf{d})$.

4.2. The "Max-Steering" distribution for quadrature filters

In this subsection, we provide an analysis similar to the previous one, but in terms of the constructed quadrature pairs.

4.2.1. The "Hyper-donut" mechanism (quadrature pairs)

The following notation is used

$$Q_{\mathbf{d}}^{\text{even}}(\boldsymbol{\omega}) := \sum_{L:\text{ even}}^{M} a_M[L](\hat{\boldsymbol{\omega}}^{\mathsf{T}} \cdot \mathbf{d})^L, \quad Q_{\mathbf{d}}^{\text{odd}}(\boldsymbol{\omega}) := \sum_{L:\text{ odd}}^{M} a_M[L](\hat{\boldsymbol{\omega}}^{\mathsf{T}} \cdot \mathbf{d})^L \quad (26)$$

to denote the quadrature pair oriented along the unit vector $\mathbf{d}$.

In correspondence to equation (21), one can show that it approximately holds:

$$\begin{aligned} &\textit{On motion hyper-plane } (\forall \boldsymbol{\omega} \perp \mathbf{n}(\mathbf{v})) : \\ &\sum_{k=1}^{K(M)} \left(Q_{\mathbf{s}_k(\mathbf{v})}^{\text{even}}(\boldsymbol{\omega}) \right)^2 = \text{const}_{\text{e}}(M), \\ &\sum_{k=1}^{K(M)} \left(Q_{\mathbf{s}_k(\mathbf{v})}^{\text{odd}}(\boldsymbol{\omega}) \right)^2 = \text{const}_{\text{o}}(M), \end{aligned} \quad (27)$$

with $\text{const}_{\text{e}}(M) \simeq \text{const}_{\text{o}}(M)$, if the number $K(M)$ is selected as in the third column of Table 2 and the vectors $\mathbf{s}_k(\mathbf{v})$ as described in the previous subsection. Equation (27) can be shown using similar arguments as in the previous subsection and a demonstration is provided in Appendix B.2.

4.2.2. Max-Steering distribution (quadrature pairs)

Similarly to equation (23), we define the "Max-Steering" distribution for quadrature pairs as follows:

$$P_Q(\mathbf{v}) := \sum_{\mathbf{x}} \sum_{k=0}^{K} \left| g_{\mathbf{s}_k(\mathbf{v})}(\mathbf{x}) \right|^2, \text{ where } g_{\mathbf{d}}(\mathbf{x}) := (q_{\mathbf{d}} * f)(\mathbf{x}) = \sum_{L=0}^{M} \alpha_M[L] \cdot y_{\mathbf{d}}^L(\mathbf{x}). \quad (28)$$

The response of the quadrature filter is denoted by $g_{\mathbf{d}}(\mathbf{x})$, while $y_{\mathbf{d}}^L(\mathbf{x})$ denotes the response of the directional filter $b_{\mathbf{d}}^L(\mathbf{x})$ of order L.

Using the fact that the even and odd filters of the pair produce real-valued and imaginary responses, respectively, (28) can be split into two parts, as follows:

$$P_Q(\mathbf{v}) = \sum_{\mathbf{x}} \sum_{k=0}^{K} \left| \sum_{L:\text{ even}} \alpha_M[L] y^L_{\mathbf{s}_k(\mathbf{v})}(\mathbf{x}) + \sum_{L:\text{ odd}} \alpha_M[L] y^L_{\mathbf{s}_k(\mathbf{v})}(\mathbf{x}) \right|^2$$
$$= \underbrace{\sum_{\mathbf{x}} \sum_{k=0}^{K} \left(\sum_{L:\text{ even}} \alpha_M[L] y^L_{\mathbf{s}_k(\mathbf{v})}(\mathbf{x}) \right)^2}_{P_Q^{\text{even}}(\mathbf{v})} - \underbrace{\sum_{\mathbf{x}} \sum_{k=0}^{K} \left(\sum_{L:\text{ odd}} \alpha_M[L] y^L_{\mathbf{s}_k(\mathbf{v})}(\mathbf{x}) \right)^2}_{P_Q^{\text{odd}}(\mathbf{v})}. \tag{29}$$

4.2.3. Steerability of $P_Q(\mathbf{v})$ and interpolation formula

It is shown here that $P_Q(\mathbf{v})$ can be interpolated from a fixed set of quadratic measurements. From the definition of $P_Q^{\text{even}}(\mathbf{v})$ in (29), and using the interpolation formula (in the original space-time domain), it holds:

$$P_Q^{\text{even}}(\mathbf{v}) = \sum_{\mathbf{x}} \sum_{k=0}^{K} \sum_{L_1,L_2:\text{ even}} \alpha_M[L_1]\, \alpha_M[L_2]\, y^{L_1}_{\mathbf{s}_k(\mathbf{v})}(\mathbf{x})\, y^{L_2}_{\mathbf{s}_k(\mathbf{v})}(\mathbf{x})$$
$$= \sum_{k=0}^{K} \sum_{L_1,L_2:\text{ even}} A_M[L_1,L_2] \sum_{i=0}^{I(L_1)} \sum_{j=0}^{I(L_2)} T_{i,j}^{\{L_1,L_2\}}\big(\mathbf{s}_k(\mathbf{v})\big) \cdot z_{i,j}^{\{L_1,L_2\}}, \tag{30}$$

where

$$A_M[L_1,L_2] = \alpha_M[L_1] \cdot \alpha_M[L_2], \quad T_{i,j}^{\{L_1,L_2\}}(\mathbf{d}) := t_i^{L_1}(\mathbf{d}) \cdot t_j^{L_2}(\mathbf{d}) \tag{31}$$

and $z_{i,j}^{\{L_1,L_2\}} := \sum_{\mathbf{x}} y^{L_1}_{\mathbf{d}_i}(\mathbf{x}) \cdot y^{L_2}_{\mathbf{d}_j}(\mathbf{x})$.

Exactly the same analysis can be performed for $P_Q^{\text{odd}}(\mathbf{v})$. Taking into account the fact that $z_{i,j}^{\{L_1,L_2\}} = z_{j,i}^{\{L_1,L_2\}}$ only when $L_1 = L_2$, the total number of quadratic measurements is

$$\sum_{L_1,L_2:\text{Even}}^{M} J(L_1,L_2) + \sum_{L_1,L_2:\text{Odd}}^{M} J(L_1,L_2), \quad \text{where}$$
$$J(L_1,L_2) = \begin{cases} \frac{I_0(L)\cdot\left(I_0(L)+1\right)}{2} & \text{if } L_1 = L_2 = L \\ I_0(L_1) \cdot I_0(L_2) & \text{if } L_1 \neq L_2. \end{cases} \tag{32}$$

4.3. Definition of voxelwise "Max-Steering" distributions

The "Max-Steering" objective functions in subsections 4.1 and 4.2 - equations (23) and (28), respectively - have been defined for the whole space-time function $f(\mathbf{x})$. Notice that the summation along $\mathbf{x}$ was for the whole space-time domain. In other words, the "Max-Steering" functions, as defined so far, could serve for the estimation of a single flow vector that describes the motion of a globally translating scalar function. Practically however, in the flow estimation problem, one is interested in estimating a dense flow-field, i.e. flow vector for each space-time location $\mathbf{x}$.

In correspondence to equation (23), one could define the distribution $P(\mathbf{x};\mathbf{v}) = \sum_{k=1}^{K} \left| y_{\mathbf{s}_k(\mathbf{v})}(\mathbf{x}) \right|^2$, i.e. drop the the summation along $\mathbf{x}$. This "voxelwise" distribution could serve for dense flow estimation at each space-time location $\mathbf{x}$. However, in practice, in order to gain robustness against noise and override the "blank-wall" problem at "textureless" regions (regions with poor power spectrum), it is preferable to locally integrate the objective function measurements. Therefore, the following voxelwise distribution is defined for the needs of the proposed algorithmic developments:

$$P(\mathbf{x};\mathbf{v}) := \sum_{\mathbf{x}_n \in \aleph(\mathbf{x})} W(\mathbf{x}-\mathbf{x}_n) \sum_{k=1}^{K} \left| y_{\mathbf{s}_k(\mathbf{v})}(\mathbf{x}_n) \right|^2, \tag{33}$$

where $\aleph(\mathbf{x})$ denotes a small space-time neighborhood around $\mathbf{x}$ and $W(\mathbf{x})$ a N-D smooth window function, centered around zero. The described modification in the definition of the "Max-Steering" distribution, does not affect the mathematical analysis in previous subsections. Therefore, using the interpolation formula of (25), $P(\mathbf{x};\mathbf{v})$ can be interpolated from a fixed set of quadratic terms of the form

$$z_{i,j}(\mathbf{x}) := \sum_{\mathbf{x}_n \in \aleph(\mathbf{x})} W(\mathbf{x}-\mathbf{x}_n)\; y_{\mathbf{d}_i}(\mathbf{x}_n)\; y_{\mathbf{d}_j}(\mathbf{x}_n). \tag{34}$$

Using the same arguments, in correspondence to equation (28), one can define the "Max-Steering" distribution for quadrature pairs:

$$P_Q(\mathbf{x};\mathbf{v}) := \sum_{\mathbf{x}_n \in \aleph(\mathbf{x})} W(\mathbf{x}-\mathbf{x}_n) \sum_{k=0}^{K} \left| g_{\mathbf{s}_k(\mathbf{v})}(\mathbf{x}_n) \right|^2, \tag{35}$$

which can be interpolated from the fixed set of quadratic terms

$$z_{i,j}^{\{L_1,L_2\}}(\mathbf{x}) := \sum_{\mathbf{x}_n \in \aleph(\mathbf{x})} W(\mathbf{x}-\mathbf{x}_n)\; y_i^{L_1}(\mathbf{x}_n)\; y_j^{L_2}(\mathbf{x}_n). \tag{36}$$

4.4. Additional issues

4.4.1. Necessity of pre-filtering

It is well-known that natural input signals, such as 2D images or 3D volumes, present low-frequency characteristics, i.e. the largest portion of their energy is concentrated in low-frequency components. Following the theoretical analysis of section 2, this means that $|F_0(\boldsymbol{\omega}_s)|$ will contain mainly low-frequency components. Therefore, the energy in the spatiotemporal FT of the input sequence $F(\boldsymbol{\omega})$, although distributed along a "motion hyper-plane", it will be mainly concentrated around DC (zero frequency). In other words, the "motion hyper-plane" will be "weak" and its detection may be problematic. Therefore, the enhancement of medium- and high-frequency components can assist the proposed methodology. For this reason, in the algorithms presented in the next section, a band-pass pre-filtering step is applied. More specifically, a pre-filter of the form $G_b(\boldsymbol{\omega}) = G(\boldsymbol{\omega})G_h(\boldsymbol{\omega}_s)$ is applied to the input sequence, where

$$G(\boldsymbol{\omega}) = \exp\left(\boldsymbol{\omega}^{\mathsf{T}}\, \Sigma\, \boldsymbol{\omega}\right) \tag{37}$$

is a N-D low-pass separable Gaussian filter, i.e. with a diagonal covariance Σ, whereas $G_h(\boldsymbol{\omega}_s) = ||\boldsymbol{\omega}_s||$ is a (hyper-)cylindrical high-pass spatial derivative-like filter. In the above definitions, the frequency vector $\boldsymbol{\omega}$ is considered normalized in the interval $[-1, 1]^N$. The use of such a pre-filter is in accordance with [3], where the 2D flow estimation task is handled. Additionally, it is generally in accordance with the 2D flow estimation literature, where the medium frequency components (edges, corners, etc.) are assumed to contain the most descriptive information for motion.

5. Algorithmic developments

In this section we describe the proposed algorithm for 3D flow estimation in sequences of volume data, based on the theoretical developments of previous sections for $N = 4$. The described algorithm is the simplest one that can be formulated based on our theoretical developments, since our major objective is to show-case the validity and usefulness of the developed theory. In subsection 5.2, a few more sophisticated alternatives for specific algorithmic parts are given.

Input: A sequence of discrete volumetric data, i.e a 4D scalar function $f(\mathbf{x})$ constitutes the input of the proposed algorithm, where $\mathbf{x} = [\mathbf{x}_s^{\mathsf{T}}; t]^{\mathsf{T}} = [x, y, z, t]^{\mathsf{T}}$ is the discrete 4D space-time vector.

Output: The output of the algorithm is a flow field $\mathbf{v}(\mathbf{x}_s)$, which is assigned to the middle sequence's frame $t = N_t/2$, where N_t is the number of input frames.

We additionally describe how the method is applied to sequences of 3D point-clouds.

5.1. 3D Flow estimation algorithm

The overall algorithm is split into a number of sequentially applied algorithmic parts, described in the subsequent paragraphs 5.1.1-5.1.5.

5.1.1. Algorithm #A0: Preparation (off-line)

Given any sequence of specific volume size and number of input volume frames, this algorithmic part (preparation) has to be performed only once. It can be summarized as follows.

Input information: a) Size $N_x \times N_y \times N_z$ of input volumes and number of input frames N_t; b) Filters order L (for simple directional filters) or M (for quadrature filters).

Algorithmic steps:

1. *Construction of pre-filters:* Based on subsection 4.4.1, construct a band-pass pre-filter $G_b(\boldsymbol{\omega})$. The diagonal Gaussian covariance matrix $\Sigma = \text{diag}\left[\sigma_{\omega_x}, \sigma_{\omega_y}, \sigma_{\omega_z}, \sigma_{\omega_t}\right]$ in equation (37) is selected with $\sigma_{\omega_x} = \sigma_{\omega_y} = \sigma_{\omega_z}$. Considering the frequency normalized in [-1,1], appropriate values used in all our experiments are $\sigma_{\omega_x} = 0.7$ and $\sigma_{\omega_t} = 1$.

2. *Construction of steerable basis:* Construct a 4D steerable filter basis of order L, in the frequency domain, i.e. filters $B_{\mathbf{d}_i}(\boldsymbol{\omega})$ at the basic orientations $\mathbf{d}_i, i = 1, 2, \ldots I_0(L)$. The minimum number of required basic filters is given by (5). The selection of basic filters orientations is discussed in subsection 3.1.2.

3. Given the basic filters orientations $\mathbf{d}_i$, calculate and store the matrix $\mathtt{U}^{-1}$, according to equations (7) and (9).

4. If quadrature filters of order M are going to be used, repeat steps 2 and 3 for all $L \leq M$. Additionally, consider the necessary coefficients α_M, given in Table 1.

5. Find a collection of K unit vectors $\mathbf{q}_k$, which are "uniformly" distributed on the sphere. The appropriate number $K(L)$ (or $K(M)$) of such vectors is given in the third column of Table 2. We solve the "hard-spheres problem" (best packing on the sphere) [27] to "uniformly" distribute points on the sphere, as explained in subsection 3.1.2.

5.1.2. Algorithm #A1: Pre-filtering and initial computations

This algorithmic part consists of the computation of a number of convolutions. All necessary convolutions are computed in the 4D spatiotemporal FT domain as products. A high computational gain is achieved with this, due to the existence of very fast FFT implementations [31, 32]. Notice that frequency-domain product corresponds to circular convolution. To handle this finite-size boundary effect, extension of the input volume boundaries could be employed (e.g. zero-padding or repetition of boundary voxels), as in [3]. However, since the spatial extent of the proposed directional filters is actually small, practically this step is omitted. The presented experimental results were obtained without such boundary handling.

Input: A sequence of discrete volume data $f(\mathbf{x}) = f(\mathbf{x}_s; t)$.

Algorithmic steps:

1. Take the 4D spatiotamporal FT of the input $f(\mathbf{x})$, to obtain $F(\boldsymbol{\omega})$, i.e. the input in the spatiotemporal frequency domain.

2. Apply the 4D band-pass filter $G_b(\boldsymbol{\omega})$ to enhance medium-frequency components: $F(\boldsymbol{\omega}) \leftarrow G_b(\boldsymbol{\omega})F(\boldsymbol{\omega})$.

3. Multiply $F(\boldsymbol{\omega})$ with each basic filter $B_{\mathbf{d}_i}(\boldsymbol{\omega})$, to obtain the basic responses $Y_{\mathbf{d}_i}(\boldsymbol{\omega}) = B_{\mathbf{d}_i}(\boldsymbol{\omega})F(\boldsymbol{\omega})$, $i = 1, 2, \ldots, I_0(L)$.

4. Apply 4D IFT to each $F_i(\boldsymbol{\omega})$ to get the basic responses $y_{\mathbf{d}_i}(\mathbf{x}) = b_{\mathbf{d}_i}(\mathbf{x}) * f(\mathbf{x})$ in the original space-time domain.

5. If quadrature filters of order M are going to be used, repeat steps 3 and 4 for all orders $L \leq M$.

5.1.3. Algorithm #A2: Calculation of "voxelwise" quadratic terms

Algorithmic steps:

Case A - Usage of simple filters:

- For a specific filters' order L, calculate the "voxelwise" quadratic terms $z_{i,j}(\mathbf{x}_s)$, defined in (34), for all combinations of $i, j = 1, 2, \ldots, I_0(L)$. The total number of quadratic terms is given by equation (24).

  Notice that the notation $z_{i,j}(\mathbf{x})$ in (34) has changed here to $z_{i,j}(\mathbf{x}_s)$. The reason is that the quadratic terms are calculated for each voxel $\mathbf{x}_s$ but only for $t = N_t/2$, since we are interested in estimating a motion field only for the central frame.

Case B - Usage of quadrature filters:

- Calculate the "voxelwise" quadratic terms $z_{i,j}^{\{L_1,L_2\}}(\mathbf{x}_s)$, defined in (36), for all combinations of even orders $L_1, L_2 = 0, 2, \ldots, M$ and combinations of odd orders $L_1, L_2 = 1, 3, \ldots, M$. For the total number of the quadratic terms, see equation (32).

Spatiotemporal integration window: The calculations of $z_{i,j}(\mathbf{x}_s)$ or $z_{i,j}^{\{L_1,L_2\}}(\mathbf{x}_s)$ in equations (34) or (36), respectively, involve the spatiotemporal integration window $W(\mathbf{x})$. In this work we use a spatiotemporal 4D Gaussian window of size $W_x \times W_y \times W_z$ voxels times N_t frames. For all our experiments, the Gaussian standard deviation is set equal to $0.5 \cdot [W_x, W_y, W_z]$ in space, and equal to $0.2N_t$ in time. In the presented experiments, the selection of window's spatial size is done intuitively, based on the size of the voxels. For example: a) In the experiment with medical MRI data of subsection 6.2.2, where the voxel size is $2.74 \times 2.74 \times 4$ mm^3, the integration window size is selected equal to $11 \times 11 \times 5$ voxels that corresponds to a region of approximate size $30 \times 30 \times 20$ mm^3; b) For the real-world human motion sequence in subsection 6.3.2, where the voxel size is $2 \times 2 \times 2$ cm^3, the integration window size is set to $11 \times 11 \times 9$ voxels that corresponds to a region of approximate size $22 \times 22 \times 18$ cm^3.

5.1.4. Algorithm #A3: Initial flow-field estimation by sparse grid search

In this algorithmic step, an initial coarse flow-field is extracted. Using brute search on a sparse 3D grid of candidate velocities $\mathbf{v}$, the velocity that maximizes the "Max-Steering" objective function $P(\mathbf{x}_s; \mathbf{v})$ in (33) (or $P_Q(\mathbf{x}_s; \mathbf{v})$ in (35)) is found for each voxel $\mathbf{x}_s$. The selection of the 3D velocity search grid is based on: a) if exists, any available information for the minimum and maximum expected speeds; b) the search step should be at most 0.5 voxels/frame (i.e. to have at least half-voxel estimation accuracy). A

smaller search step is preferable, to ensure that the found maximum is close to the global maximum of $P(\mathbf{x}_s; \mathbf{v})$. On the other hand, since brute search in the 3D space is computationally expensive, one should avoid using dense search grids. Fortunately, in practice, the function $P(\mathbf{x}_s; \mathbf{v})$ is quite regular, in the sense that it does not present several and complex local minima.

Input information: A sparse 3D grid of candidate velocities.

Algorithmic steps:

For each candidate velocity $\mathbf{v}$:

1. Find the vectors $\mathbf{s}_k(\mathbf{v}), k = 1, 2, \ldots, K$ according to the analysis of subsection 4.1 and more specifically using equation (20). The collection of the K unit vectors $\mathbf{q}_k$, precalculated in the last step of Algorithm #0, is used.

2. For each $k = 1, 2, \ldots, K$, calculate the interpolation vectors $\mathbf{t}\big(\mathbf{s}_k(\mathbf{v})\big)$ according to the analysis of subsection 3.1.2.

3. Compute the interpolation coefficients $T_{i,j}\big(\mathbf{s}_k(\mathbf{v})\big) = t_i\big(\mathbf{s}_k(\mathbf{v})\big) \cdot t_j\big(\mathbf{s}_k(\mathbf{v})\big)$. If quadrature filters of order M are used, calculate the interpolation coefficients $T_{i,j}^{\{L_1,L_2\}}\big(\mathbf{s}_k(\mathbf{v})\big)$, for all combinations of even orders $L_1, L_2 = 0, 2, \ldots, M$ and combinations of odd orders $L_1, L_2 = 1, 3, \ldots, M$. Additionally, compute the coefficients $A_M[L_1, L_2] = \alpha_M[L_1] \cdot \alpha_M[L_2]$.

4. Compute the "voxelwise" "Max-Steering" objective function

   - *Case A - Usage of simple filters*: $P(\mathbf{x}_s; \mathbf{v})$ which is defined in (33), using the interpolation formula of equation (25).
   - *Case B - Usage of quadrature filters*: $P_Q(\mathbf{x}_s; \mathbf{v})$ which is defined in (35), using the interpolation scheme of (30).

For each voxel $\mathbf{x}_s$:

5. Find which candidate velocity $\mathbf{v}$ minimizes the objective function $P(\mathbf{x}_s; \mathbf{v})$ (or $P_Q(\mathbf{x}_s; \mathbf{v})$ for quadrature filters).

Output: An approximate 3D flow-field $\widehat{\mathbf{v}}(\mathbf{x}_s)$.

5.1.5. Algorithm #A4: Final flow-field estimation by downhill simplex optimization

Given the initially estimated 3D flow-field of the previous algorithmic part, here the flow-field is refined towards high accuracy, by the use of iterative downhill simplex optimization [33, ch. 10]. In other words, for each voxel $\mathbf{x}_s$, the exact maximum of $P(\mathbf{x}_s; \mathbf{v})$ (or $P_Q(\mathbf{x}_s; \mathbf{v})$ for quadrature filters) is iteratively searched, beginning from the initial estimate.

Input information: An approximate 3D flow-field $\widehat{\mathbf{v}}(\mathbf{x}_s)$.

Algorithmic steps:

For each voxel $\mathbf{x}_s$:

- Starting from $\widehat{\mathbf{v}}(\mathbf{x}_s)$, iterate using downhill simplex optimization towards the minimization of $-P(\mathbf{x}_s; \mathbf{v})$ (or $-P_Q(\mathbf{x}_s; \mathbf{v})$ for quadrature filters). In each iteration (and for each simplex vertex [33, ch. 10], i.e. for each examined velocity) the steps 1-4 of Algorithm #A3 are executed.

Output: The final estimated 3D flow-field $\mathbf{v}(\mathbf{x}_s)$.

Providing details of the downhill simplex optimization method (or Nelder-Mead method) is beyond the scope of this paper and the reader is referred to [33, ch. 10]. With respect to the optimization parameters used in our experiments, we mention that the initial simplex size is 0.1, the tolerance for termination is 10^{-5} and the maximum number of iterations is 500.

We note here that there is no special reason for choosing the Nelder-Mead method in this algorithmic step. As stated, in practice, the "Max-steering" distribution is quite regular, in the sense that it does not present several and complex local minima, and smooth. We expect that the use of other faster local optimization approaches, such as gradient-descent or Levenberg-Marquardt, would produce similar results, since the initial estimates provided by Algorithm #A3 are close to the actual velocities.

5.2. Algorithmic alternatives

In Algorithm #A3 one has to ensure that the search grid is dense enough on one hand and on the other hand the search limits are adequately large. This means that the number of candidate velocities (size of search grid) may be quite large, resulting into high memory and computational requirements. Therefore, an efficient alternative to the Algorithmic step #A3 would be

the employment of global stochastic optimization methods, e.g. particle-based sequential Monte-Carlo methods [34] such as Interacting Simulated Annealing [35].

The spatiotemporal summation in equations (34) or (36) involve an integration window, which is fixed and independent to the local volume content (Algorithm #A2). According to the "generalized aperture problem" [14], the window ("aperture") on one hand has to be adequately large in order to constrain motion and on the other hand it should be small enough, so that not to contain multiple motions; otherwise, estimation errors or flow-field over-smoothing may occur at object boundaries (motion discontinuities). To override such problems, local content-based adaptive windows could be used or guided image (volume in our case) filtering ideas [36].

5.3. Application to 3D point-cloud sequences

To use sequences of 3D point clouds in our underlying 3D flow estimation framework, a volumetric function has to be constructed from each input point cloud. There are various ways to realize this. For example, in various 3D surface reconstruction methods [37, 38], a volume function $F(\mathbf{x}_s)$ is generated from a set of input point-normals. This function is (almost) zero near the surface implicitly defined by the input points, negative outside and positive inside the surface. In other words, each voxel is assigned a value that depends on its distance from the actual 3D surface, imposed by the input point cloud.

However, in this work, we preferred to use the simplest possible approach. A binary (0/1) volumetric function is constructed, with zeros and ones indicating empty and occupied voxels, respectively. This simple scheme, apart from keeping the proposed method as simple as possible and therefore showcases the validity of our arguments, keeps the requirements on the input to their minimum. For example, this approach does not need the calculation of the 3D surface normals and obviously does not need connectivity (mesh) information.

5.3.1. Algorithm #B: 3D flow estimation in sequences of point-clouds

Input: A sequence of N_t consecutive point clouds, where each point-cloud is a set of $N(t)$ points $\mathbf{p}_n(t), n = 1, 2, \ldots, N(t)$. This means that the number of input points can very for frame to frame.

Input information: Wanted size of the voxels. In the experiments of subsection 6.3, the voxel size is selected based on the moving object's size and

its details. For example, for the real-world sequences with a moving human in subsection 6.3.2 (approx. height 180cm), the voxel size is selected equal to $2 \times 2 \times 2$ cm^3. Whereas, for the artificial "Armandillo" sequence (see subsection 6.3.1), which was generated from a small creature object (of approx. height 180mm), the voxel size is selected much smaller, equal to $2 \times 2 \times 2$ mm^3.

Output: A flow field $\mathbf{v}(\mathbf{p}_n)$ on a per-point basis, i.e. a velocity vector for each input point $\mathbf{p}_n$. Notice again that the output estimated filed is assigned to the central frame of the sequence $t = N_t/2$.

Algorithmic steps:

1. The 3D bounding box for all input point clouds is initially found. The bounding box is then uniformly discretized into $N_x \times N_y \times N_z$ cubic voxels, based on the wanted voxel size.

2. Iterating for all input points $\mathbf{p}_n(t)$ a voxel is indicated as "occupied (i.e. $F(\mathbf{x}_s; t)$ is set to unity) if it contains at least one 3D point, otherwise it is indicated as "empty" (i.e. $F(\mathbf{x}_s; t)$ remains zero). Additionally, in this step, we keep track of the voxel $\mathbf{x}_s$ to which a point $\mathbf{p}_n(t)$ belongs in. Formally, a mapping $\mathbf{p}_n(t) \leftrightarrow \mathbf{x}_s[n; t]$ is constructed.

3. Apply the overall Algorithm #A, described in the previous subsection 5.1, in order to estimate the voxelwise flow-field $\mathbf{v}(\mathbf{x}_s)$. However, the algorithmic parts #A3, #A4 and #A5 should be executed only for the "occopied" voxels. With respect to #A2, the "voxelwise" quadratic terms $z_{i,j}(\mathbf{x}_s)$ can be calculated only at the neighborhoods $\aleph(\mathbf{x}_s)$ of each "occopied" voxel, as dictated by (34).

4. Having the mapping $\mathbf{p}_n(t) \leftrightarrow \mathbf{x}_s[n; t]$, each point $\mathbf{p}_n(t)$ for $t = N_t/2$ is assigned the corresponding flow-vector $\mathbf{v}(\mathbf{x}_s)$. This results into the output pointwise flow-field $\mathbf{v}(\mathbf{p}_n)$.

We note here that there are more sophisticated approaches to perform the last step. For example, the flow-vectors assigned to a 3D point $\mathbf{p}_n$ could be calculated as the weighted average of the estimated flow-vectors $\mathbf{v}(\mathbf{x}_s)$ in the neighborhood of $\mathbf{p}_n$ and using Euclidean distance-based weights. However, as mentioned, we prefer to keep the proposed algorithms as simple as possible.

5.4. Notes on the computational effort and memory requirements

The computational effort of the proposed approach depends mainly on three parameters: a) The filters' order L; b) the size of the integration window $\aleph(\mathbf{x}_s)$, used in (34) and c) obviously the input volume size. The most time-consuming algorithmic part of the overall approach is Algorithm #A2, which requires the calculation of a number of "voxel-wise" quadratic terms, as in (34). The number $J(L)$ of these terms is given from (24) and it is a quadratic function of $I_0(L)$, which is given in the 2nd column of Table 2. Namely, we have $J(1) = 10$, $J(2) = 55$ and $J(3) = 210$ quadratic terms for L=1, 2 and 3 respectively, and so on. Additionally, the computational effort for calculating each "voxel-wise" quadratic term is linearly proportional to the size of the integration window, while the size of each term is equal to the size of the input volume.

The effort for steps Algorithm #A3 and Algorithm #A4, which target the calculation of "Max-steering distribution" for each candidate velocity, is proportional to the number of directions $K(L)$ (see eq. (33)), which is given in the 3rd column of Table 2. Thus, the effort for these parts are also affected by filter's order L. Finally, specifically considering Algorithm #A3, its computational time is proportional to the size $N_U = N_{Ux} \times N_{Uy} \times N_{Uz}$ of the used sparse 3D search-grid of velocities. Actually, Algorithm #A3 is not time-consuming, compared to Algorithm #A2 and Algorithm #A4; thus the overall approach is only slightly affected by the size of this search-grid.

With respect to the memory requirements, these are high. Specifically, the storage of $J(L)$ "voxelwise" quadratic terms is needed, each of size equal to that of the input volume. Additionally, in our implementation N_U "vowelise Max-steering distributions" have to be stored. Thus, large volume sizes and/or filters' order and/or dense search grids are practically prohibitive. This constitutes a practical limitation of the proposed algorithm, given that a straightforward implementation is used.

The above analysis stands also when using quadrature pairs. Coarsely, based on the theoretical analysis, the use of quadrature filters doubles the computation effort and the memory requirements.

Finally, as explained in Algorithm #B (subsection 5.3), when the approach is applied to point-clouds, the algorithmic parts #A3, #A4 and #A5 are executed only for the "occupied" voxels. Thus, the computational effort is reduced. Additionally, the effort could be reduced in #A2, as explained, calculating the quadratic terms only at the neighborhoods of the "occupied"

voxels. This could also reduce the memory requirements. However, our implementation does not take advantage of the above fact.

6. Experimental Results

In this section, we present the experimental results of the proposed method on both pure volumetric data (subsection 6.2) and 3D point-cloud data (subsection 6.3). Evaluation is performed quantitatively when the ground-truth (GT) is known, qualitatively otherwise. Apart from estimation accuracy, the computational performance is also investigated. The proposed algorithm was implemented in C++. The reported execution times refer to a laptop PC, with an Intel(R) Core (TM) i7-4510U CPU, at 2.00GHz, 16GB RAM and a 64-bit operating system.

Appropriate values for most of the proposed algorithm's parameters and commonly used values for all the experiments were given in section 5. Values for other parameters are given separately for each experiment. The proposed algorithm, is fed with N_t=5, 6 or 7 consecutive frames and the estimated motion field is associated with the central frame.

Comparative results are also given using the 3D extension [15] of the well-known Lucas-Kanade (LK) method [13], which is shortly described in paragraph 6.1.1 below.

6.1. Prerequisites

6.1.1. The 3D Lucas-Kanade method

We implemented and experimented with the 3D extension of the LK flow estimation method [13], as this is described in [15]. Briefly, the velocity is estimated by minimizing the function: $\sum_{\mathbf{x}_n \in \aleph(\mathbf{x}_s)} W^2(\mathbf{x}_s - \mathbf{x}_n)\big(F_{\mathbf{x}}(\mathbf{x}_n)^{\mathrm{T}} \cdot \mathbf{v} + F_t(\mathbf{x}_n)\big)$, where $F_{\mathbf{x}}(\mathbf{x}_n) = \nabla F(\mathbf{x}_n)$ is the spatial gradient and $F_t(\mathbf{x}_n)$ the temporal derivative of the volume sequence $F(\mathbf{x}_s; t)$ at the frame of interest. $\aleph(\mathbf{x}_s)$ denotes a cubic neighborhood around voxel $\mathbf{x}_s$ and $W(\mathbf{x}_s)$ a 3D Gaussian window. The standard deviation of this window was set 0.5 times the window's radius in all the experiments. The velocity is estimated by setting up and solving (in the least-square sense) an over-determined system of linear equations. The differentiations for the calculation of spatial and temporal gradients was preformed using Simoncellis derivative filters [39], as suggest in [15]. The method was implemented in C++ using Eigen[1], a library

[1] `http://eigen.tuxfamily.org/`

optimized for matrix operations, e.g. for solving linear systems. As with the proposed method, in the case of point-clouds as input, a sequence of binary volumes is constructed and the method is applied only for the "occupied" voxels.

6.1.2. Evaluation metric

When the GT flow field is known, the Mean Angular Error MAE [13] is used as an evaluation metric, generalized for the N-D case. Given the GT velocity vector $\mathbf{v}_o$ and the corresponding estimated vector $\mathbf{v}$, the Angular Error (AE) is given from $\text{AE} = \cos^{-1}\left(\mathbf{s}(\mathbf{v})^{\mathsf{T}} \cdot \mathbf{s}(\mathbf{v}_o)\right)$, where $\mathbf{s}(\mathbf{v})$ is given from (18) and is actually the normalized homogenous-coordinates representation of $\mathbf{v}$. The MAE is obtained as the mean of Angular Errors for all motion vectors. When dealing with point-cloud sequences, the AE is calculated on a per-vertex (instead of a per-voxel) basis, since the GT motion is known for each vertex.

Additionally, although 2D slices of the estimated motion field (or the corresponding AE) are plotted, the given MAE values correspond to the field in the whole volume.

6.2. Volume sequence data

6.2.1. Artificial data

To serve the need of experimenting using data with known ground-truth (GT), three artificial volumetric sequences were generated.

Sequence #1: Uniformly translating volume: The specific sequence contains six frames and the volume size is $N_x \times N_y \times N_z = 32 \times 64 \times 16$ voxels. It consists of a single object, which translates uniformly with a velocity equal to $\mathbf{v} = [1, 1, 0]^{\mathsf{T}}$ voxels/frame. A slice $z = N_z/2$ of this translating sequence is depicted in Figure 5(a). The volumetric object is a zero-mean, spatially-uncorrelated (white) Gaussian-distributed random pattern, smoothed with a low-pass Gaussian filter $G(\boldsymbol{\omega}_s)$ with $\sigma_{\boldsymbol{\omega}_s} = [0.3, 0.3, 0.3]^{\mathsf{T}}$. In other words, the volumetric object is a spatially-correlated random 3D signal with Gaussian power spectrum.

The described "noiseless" volume sequence, is corrupted by Additive spatiotemporally White Gaussian Noise (AWGN), with controllable power (standard deviation). A frame with noise standard deviation (std) equal to 0.08 is shown in Figure 5(a). A cross-section (slice) of the motion field, obtained by applying the proposed method (with simple filters of order $L = 3$) to the

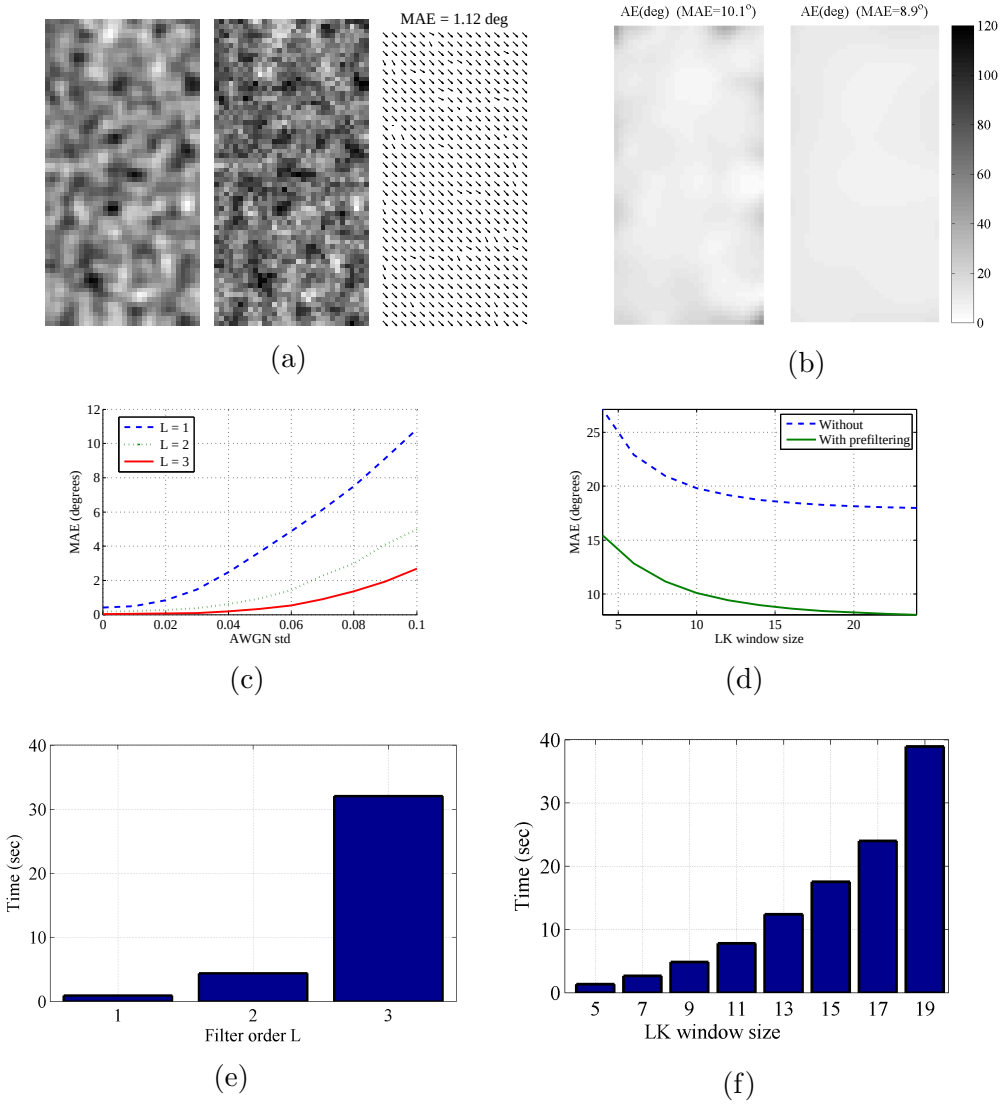


Figure 5: Sequence #1: (a) In all diagrams, a slice ($z = N_z/2$) of the data is given. From left to right: Sixth frame of the noiseless volume sequence, the same frame with AWGN of std=0.08, and the estimated flow field for the noisy sequence, using filters of order $L = 3$; (b) Angular Error (AE) of the 3D LK method for two different integration window radii (10 and 20 pixels); (c) MAE of the proposed method with various filter orders, with respect to noise power; (d) MAE of the 3D LK method for the noiseless sequence, with respect to the integration window size; (e) Execution time (C code) of the proposed method, with respect to the filters order; (f) Execution time (Matlab code) of the 3D LK method, with respect to the LK window radius.

specific noisy sequence, is given at the right of Figure 5(a). The corresponding MAE is low, equal to 1.12^o. Only the initial grid-based search part of the method is applied, using init search-grid $7 \times 7 \times 3$ and a small integration window of $3 \times 3 \times 3$ voxels.

The objective of this experiment is multi-fold. Initially, the robustness of the method against noise, with respect to the filters' order, is studied. In the diagram of Figure 5(c), the MAE error with respect to the noise level is plotted for different filters' order L. The results were obtained by carrying out 10 experiments per noise level. As can be verified, the larger the filter order the more robust the method is. For filters of order $L = 3$, the MAE errors remain below 3^o, even for strong noise.

Secondly, we study the execution time of the method with respect to the filters' order L. The mean execution time with respect to L is depicted in the diagram of Figure 5(e). For filters of order $L = 1$, the execution time is below 1sec, whereas for $L = 3$ the execution time is higher than 10sec. For relevant notes on the computational effort, the reader is referred to subsection 5.4.

Thirdly, we study the performance of the 3D LK method and perform comparisons with the proposed method. The parameter that affects the LK method is the integration window size. The blue dashed line in the diagram of Figure 5(d) shows the MAE error with respect to the radius of the integration window. The specific results are for the noiseless sequence. One can observe that the MAE of 3D LK method converges to approximately 18^o as the window radius increases. The MAE remains in high levels, regardless of the window size. The existence of high-frequency components in the moving 3D signal may affect the method's accuracy. For signals with high-frequency components, the 1st-order approximation of Taylor expansion to derive the flow constraints in the LK method, does not hold. In other words, the use of only the 1st derivatives (gradients) is not enough. Therefore, to have a more fair comparison, we applied a spatial Gaussian low-pass pre-filter in the frequency domain $G(\boldsymbol{\omega}_s)$ with $\sigma_{\boldsymbol{\omega}_s} = [0.15, 0.15, 0.15]^\intercal$. The specific values were found after trial-and-error for minimizing the MAE error. The corresponding MAE errors with respect to the integration window radius are plotted with the green line in the diagram of Fig. 5(d). As can be verified, the error becomes significantly smaller, but the LK method still performs worse than the proposed method. The distribution of the errors, i.e. the Angular Error, is given for a 2D slice in Fig. 5(b). As expected, since the velocity is constant for the whole volume, the error is almost uniformly distributed, with slightly higher errors at the boundaries when a small integration window is

used.

The computational time of the 3D LK implementation is given in the diagram of Figure 5(f) and it can be verified that it increases with the integration window size. The computational time of the proposed method is of the same order of magnitude with that of the LK method. This indicates that the proposed method is quite fast given the high amount of data in four dimensions.

Sequence #2: Two-objects volume: The sequence consists of six frames of volumes with size $32 \times 64 \times 16$ voxels. The slice $z = N_z/2$ of the middle frame is given in Figure 6(a). The sequence contains a "background" random-pattern object, with exactly the same characteristics as in the previous experiment, as well as an "occluding" ellipsoid random-pattern object (its const-z slices are circles). The X,Y,Z axes of the ellipsoid are 20, 20 and 10 voxels, respectively, while its random pattern has a power spectrum similar to that of the background. However, it has a mean value equal to 3.5 times its standard deviation. This introduces a strong "edge" at the boundaries of the two objects. The two objects move with different velocities, introducing a "strong" motion discontinuity. The ellipsoid object translates with a velocity equal to $[0, -1, 0]^{\mathsf{T}}$ upwards and the background moves with velocity $[1, 1, 0]^{\mathsf{T}}$ voxels/frames.

Applying the proposed method with quadrature filters of order $M = 3$, the obtained flow-field has a MAE equal to 6.48^o and is depicted at the left of Figure 6(b). For the specific results, down-hill optimization was not used, the initial velocity search-grid is $9 \times 9 \times 3$ and an integration window of $5 \times 5 \times 5$ voxels..As can be verified also from the illustration at the right of Figure 6(b), the estimation error is concentrated mainly at the object boundaries (motion discontinuities). For comparison, the 3D LK method is also applied with a window radius equal to 8 voxels. To assist LK method, a spatial Gaussian low-pass pre-filter was applied, with the same characteristics as in the previous experiment. The estimation results are given in Figure 6(c). The estimation error is high and mainly at the object boundaries, i.e. at the motion discontinuity. One can notice from the diagram of Figure 5(c) that a radius equal to 8 is a reasonable selection, otherwise the error would be higher at the smooth motion regions. The validity of this selection is demonstrated also from the diagram in Fig. 7(a), which shows the MAE vs the windows integration radius. Additionally, the AE for the central slice, using various integration windows, is given in Fig. 7(b).

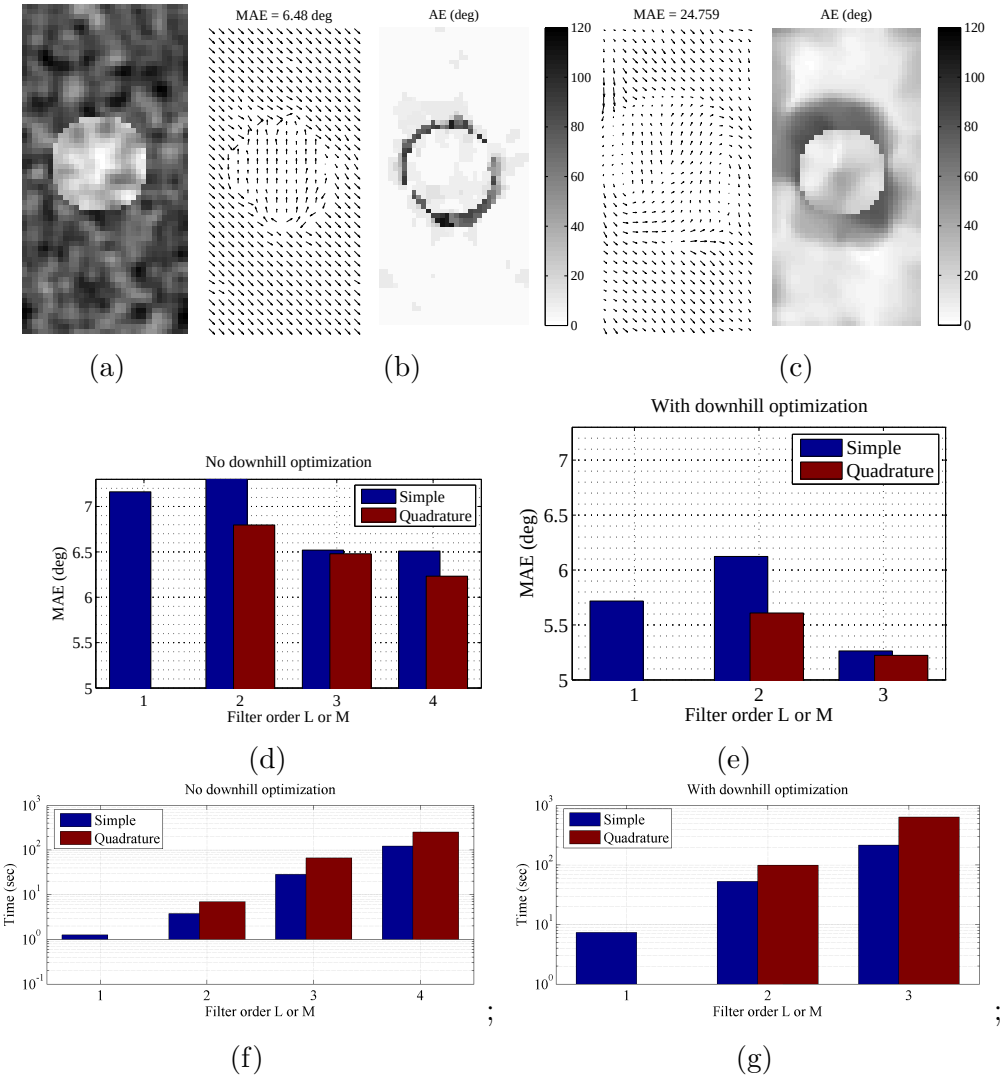
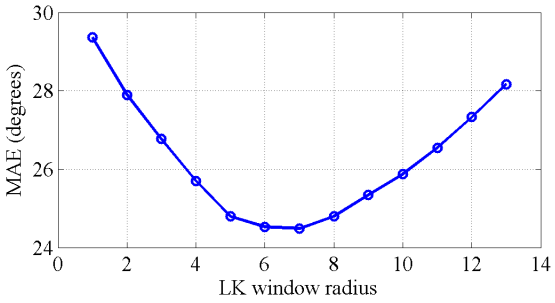
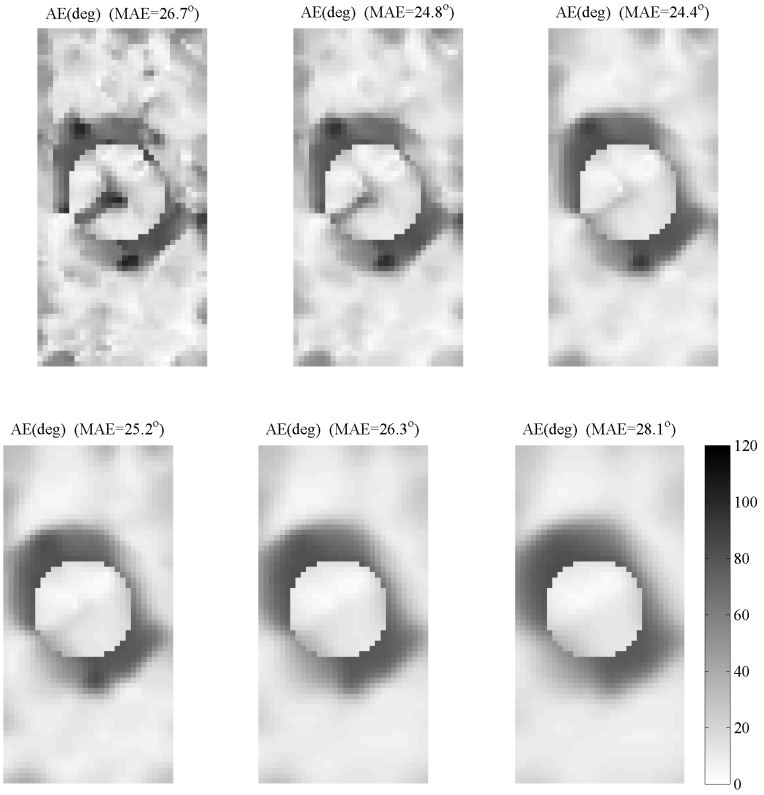


Figure 6: Sequence #2: In all diagrams, a slice ($z = N_z/2$) of the data is given. (a) Third frame of the volume sequence; b) The estimated flow field and the distribution of the error, using quadrature filters of order $M = 3$; c) The corresponding results of the 3D LK method; (d)-(g) MAE and execution time of the proposed method without and with downhill optimization.



(a) 3D Lucas-Kanade MAE with respect to integration window



(b) 3D Lucas-Kanade AE for integration windows radius 3, 5, 7, 9, 11, 13 pixels

Figure 7: Sequence #2 - Experimental results for LK method: (a) MAE with respect to integration window; (b) the corresponding error distribution for different integration windows.

The MAE of the proposed method with respect to the filter order is depicted in Figure 6(d), for both simple filters (blue bars) and quadrature pairs (red bars). A significant fact can be observed: In the previous experiment, the MAE error decreased as the filter order was increased. This does not hold in the current experiment for the simple filters case. For example, filters of order $L = 1$ perform better than filters of order $L = 2$. This can be justified by the fact that the response of the simple filters are local phase-dependent, as explained in subsection 3.2. As can be verified from the same diagram, the performance improves with the use of quadrature filters and the MAE reduces as the filters' order M increases.

The diagram of Figure 6(e) presents the experimental results after the application of the downhill optimization step. As expected, the estimation improves. The diagrams of Figure 6(f),(g) show the execution time of the algorithm with and without downhill optimization of the algorithm, respective. The scale of the vertical axis is logarithmic. The computational time is doubled with the use of quadrature pairs, as approximately expected by the analysis of subsection 4.2.3. The same diagrams reveal that unfortunately the computation effort increases significantly as the filters order goes high, as explained in subsection 5.4.

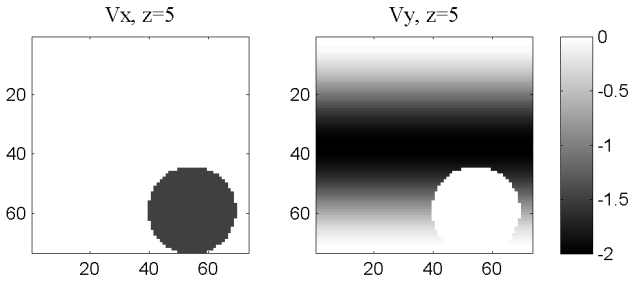
Sequence #3: "Lenna"-"Barbara" volume:

Since the LK method is affected by the existence of high-frequency components, a volume sequence was created with the use of natural images, the well-known "Lenna" and "Barbara". The sequence consists of $Nt = 7$ frames of volumes with size $73 \times 73 \times 17$ voxels. The central xy-slice ($z = 9$) for three different frames is given in Fig. 8(a), while two additional slices for the last frame are given in Fig. 8(b). As can be seen, the volume consists of a "background" object, "Lenna", and an ellipsoid "occluding" object, "Barbara". The major axes of this ellipsoid are $30 \times 30 \times 15$ pixels. While the "Barbara" object moves with a constant speed 1.5 voxels/frame towards left, the motion field of the background object is not spatially constant. It is zero at the boundaries and the U_y speed changes sinusoidally with Y, so that it is equal to 2 voxels/frame at the center. The central slice of the motion field is depicted in Fig.8(c).

The results of the proposed method are given in Figs. 9 and 10, along with comparative results. In Fig. 9(a) the results of the 3D LK method are given. Specifically, the MAE and the execution time with respect to integration window radius are given. The corresponding results of the proposed method

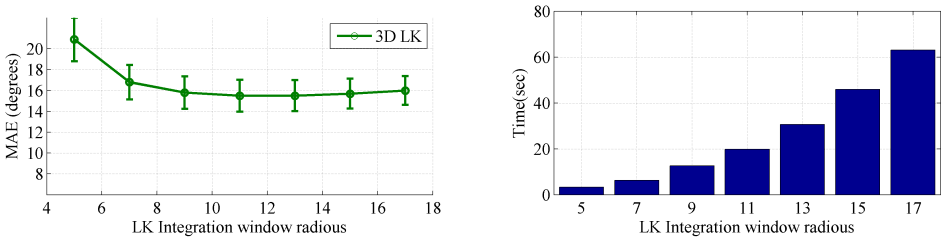
(a) Middle xy-slice ($z = 9$) at frames t=1,4,7.

(b) Two additional xy-slices (z=3 and z=5) at frame t=7.

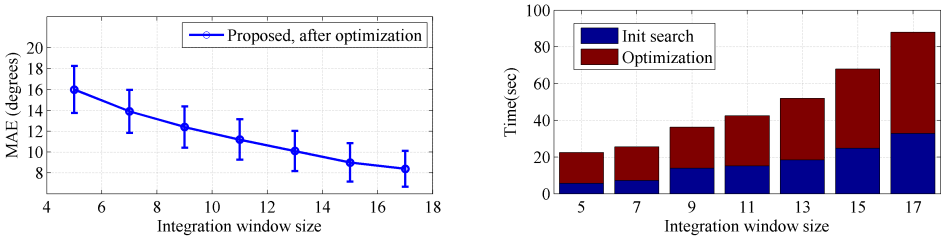


(c) Ground-Truth motion (V_x and V_y) for the middle xy-slice.

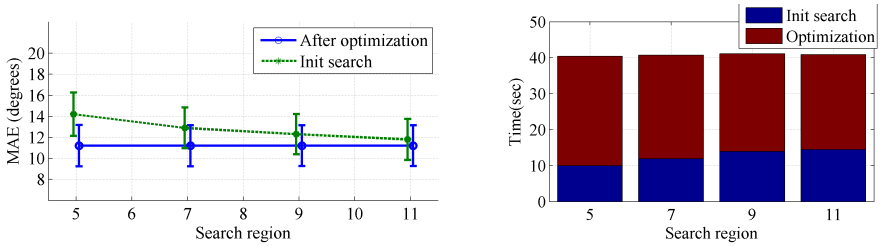
Figure 8: Sequence #3 ("Lenna"-"Barbara"): (a),(b) Slices of the volume sequence at different frames; (b) Ground-truth motion.



(a) 3D Lucas-Kanade results.

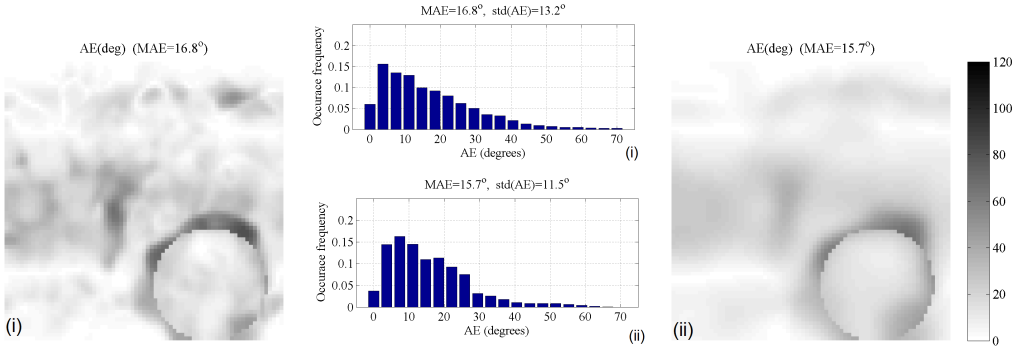


(b) Proposed method results with respect to integration window.

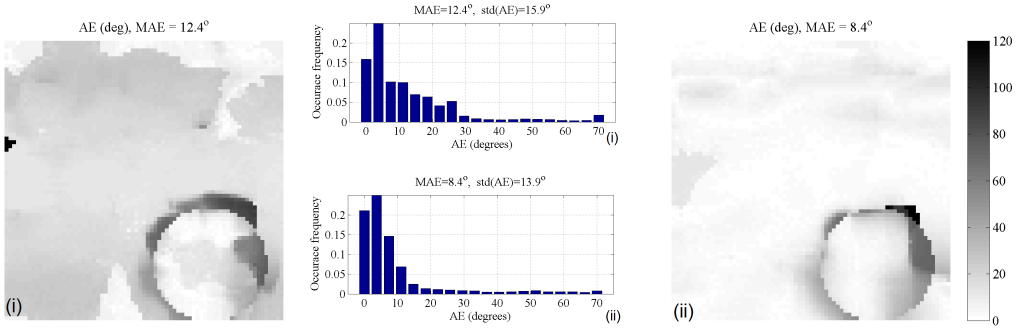


(c) Proposed method results with respect to init search region.

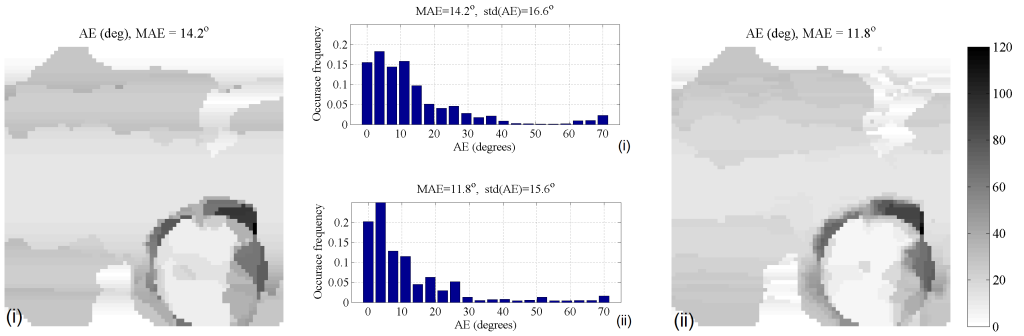
Figure 9: Sequence #3 ("Lenna"-"Barbara") - Experimental results: (a) 3D LK results - MAE and execution time vs Integration window radius; (b) Proposed method's results with respect to integration window, for search region size $9 \times 9 \times 5$; (c) Proposed method's results with respect to search region size ($N_U \times N_U \times 5$), for fixed integration window of size $11 \times 11 \times 11$ voxels. In the MAE graphs (left), the vertical lines represent 1/4 times the standard deviation of the Angular Errors.



(a) 3D Lucas-Kanade AE for different integration windows and error distributions



(b) Proposed method AE (after optimization) for different integration windows and error distributions



(c) Prop. method for different search regions (without optimization)

Figure 10: Sequence #3 ("Lenna"-"Barbara") - Experimental results: (a) AE of 3D LK method with integration window radius 7pixels (left) and 15pixels (right); (b) AE of the proposed method (after optimization) for integration window of size 9pixels (left) and 17pixels (right). The search region is $9 \times 9 \times 5$; (c) Proposed method's results for search region sizes $N_U \times N_U \times 5$, with N_U=5 (left) and N_U=11 (right). The integration window is of size $11 \times 11 \times 11$ pixels. In each subfigure, the distribution of the errors is given in the middle column.

after optimization, using low-order filters (L=1) and init search region of size $9 \times 9 \times 5$ are given in 9(b). Additionally, to study the method's accuracy with respect to the init search region, the integration window size was fixed to $11 \times 11 \times 11$ voxels and the search region $N_U \times N_U \times 5$ was varied with $N_U = 5, \ldots, 11$. The corresponding results are given in Fig. 9(c). In the left diagram of Fig. 9(c), the MAE for init search-only is also plotted. In all diagrams, the vertical lines reflect the standard deviation of the Angular Errors.

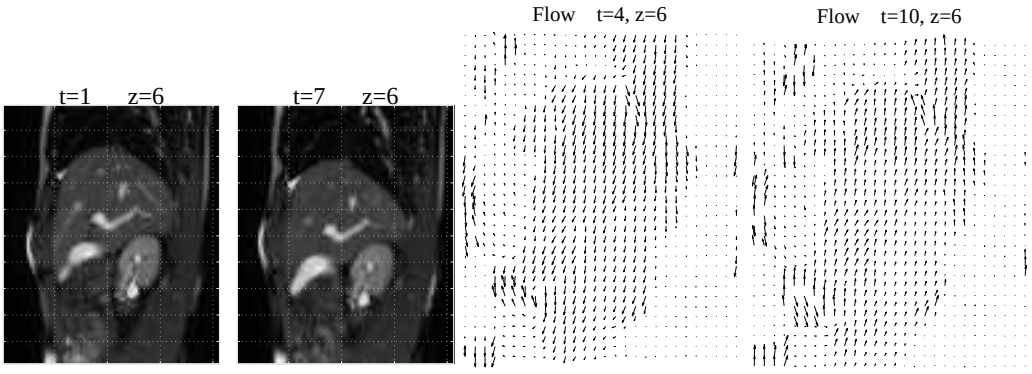
Finally, Fig. 10 depicts the distributions of the AEs, both spatially and as a probability density (histogram), for the LK method (Fig. 10(a)) and proposed one, with and without the optimization part (Figs. 10(b),(c), resp). In each case, two different integration windows are used (left and right).

The conclusions can be summarized as follows: a) With various parameters selections, in almost all cases the proposed method performs better, requiring a slightly higher computational effort; b) The computational effort increases for both methods with the integration window size; c) The init-search part of the proposed method (Algorithm #3) is affected by the selection of the search grid, but not significantly. As expected, the denser the search-grid, the more accurate and slower the algorithm is; (d) However, given that the search-grid was dense enough so that Algorithm #3 found a velocity close to the actual one, then the proposed method after Algorithm #4 (optimization) results into more-or-less the same motion-field, regardless of the density of the init search-grid. Actually, the denser the initial search-grid was, the faster the optimization algorithm converged (see Fig. 9(c)-right), since the initial solution was closer to the final one; (e) From Fig 10, it can be seen that the proposed method presents some outlier estimates (e.g. observe the histograms at the AE bin 70). Such outliers could be easier detected and removed by some post-processing algorithm.

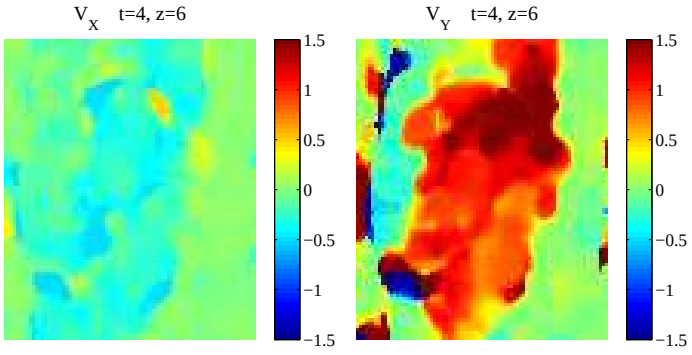
6.2.2. Real-world medical data (4D MRI)

A real-world medical data (4D MRI) sequence is used in this experiment. The sequence can be found at [40, 41] and contains 14 consecutive volumes of the liver and the gall bladder, during one breathing cycle. The temporal resolution is 362msec and each volume consists of $166 \times 195 \times 25$ voxels of size $1.37 \times 1.37 \times 4$ mm^3.

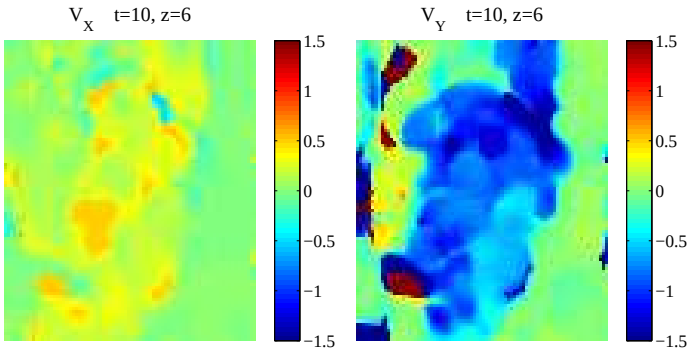
The sequence was down-sampled by a factor of 2 along X and Y, i.e. the volume resolution became $83 \times 98 \times 25$ voxels of size $2.74 \times 2.74 \times 4$ mm^3. A const-z cross-section for two frames is depicted in Figure 11(a). As can be



(a) Slice $Z = N_z/4$ (b) Proposed $t = 4$ and $t = 10$

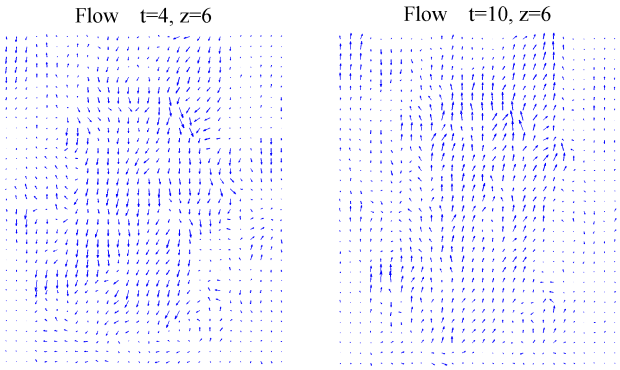


(c) Proposed $t = 4$

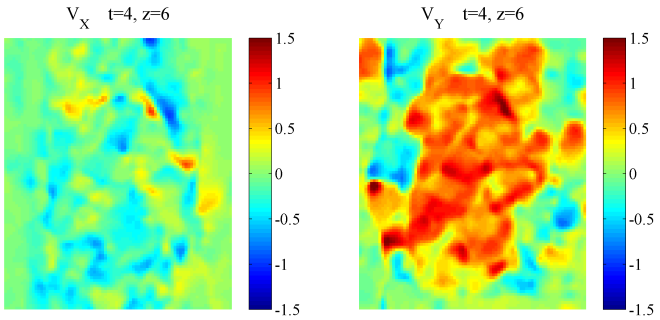


(d) Proposed $t = 10$

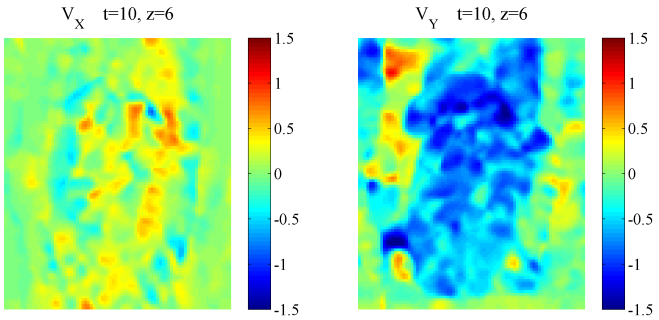
Figure 11: Sequence #4: In all diagrams, a slice ($z = N_z/4$) of the data is given. (a) Frames 1 and 7 of the volume sequence; (b) Estimated X-Y flow for $t = 4$ and $t = 10$. (c),(d) Estimated V_X, V_Y for $t = 4$ and $t = 10$. The results were obtained by the proposed method.



(a) LK $t = 4$ and $t = 10$, $(6 \times 6 \times 6)$

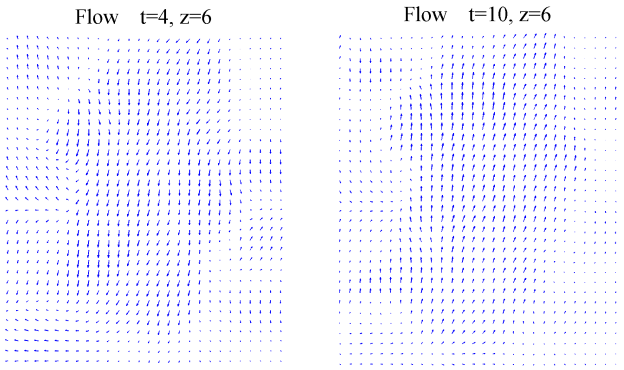


(b) LK $t = 4$, $(6 \times 6 \times 6)$

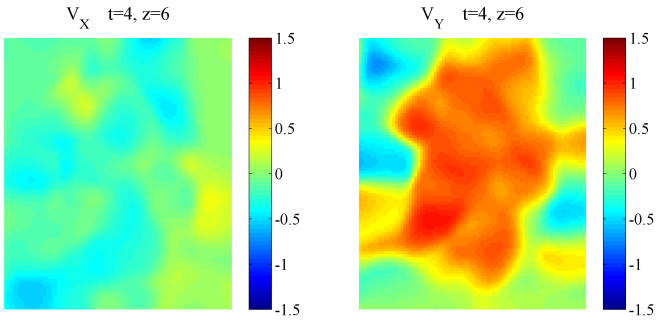


(c) LK $t = 10$, $(6 \times 6 \times 6)$

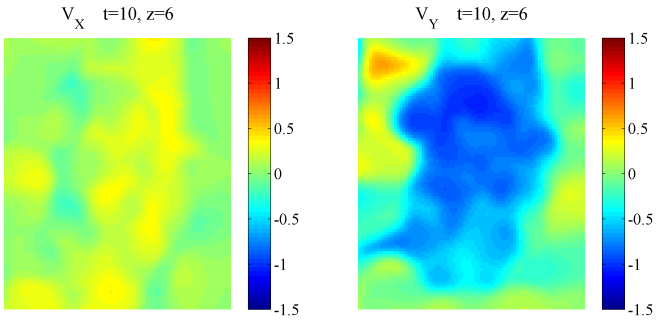
Figure 12: Sequence #4: In all diagrams, a slice ($z = N_z/4$) of the data is given. (a) Estimated X-Y flow for $t = 4$ and $t = 10$. (c),(d) Estimated V_X, V_Y for $t = 4$ and $t = 10$. The results were obtained by the LK method with an integration window size of $6 \times 6 \times 6$.



(a) LK $t = 4$ and $t = 10$, $(14 \times 14 \times 7)$



(b) LK $t = 4$, $(14 \times 14 \times 7)$



(c) LK $t = 10$, $(14 \times 14 \times 7)$

Figure 13: Sequence #4: In all diagrams, a slice ($z = N_z/4$) of the data is given. (a) Estimated X-Y flow for $t = 4$ and $t = 10$. (c),(d) Estimated V_X, V_Y for $t = 4$ and $t = 10$. The results were obtained by the LK method with an integration window size of $14 \times 14 \times 7$.

seen, the occupied part (not completely black) of the volume function moves mainly towards bottom and slightly towards left until frame 7. At frame 8 the dominant motion changes direction and the volume moves towards its initial position until frame 14, where a new cycle begins.

The proposed methodology was applied using $N_t = 6$ consecutive frames and the estimated field is assigned to the middle frame of each subsequence. Simple filters of order $L = 2$ are used and the downhill optimization step is employed. The algorithm required 56 sec for the initial velocity search (a search grid of size $5 \times 7 \times 3$ was used) and 277 additional seconds for downhill optimization. The computation time is high, but reasonable given the large volume size and the filter order $L = 2$.

The flow estimation results of the proposed method for frames t=4 and t=10 are given in Figure 11(b)-(d), respectively. Although quantitative analysis cannot be performed, qualitatively the results are sensible. It seems that the method "captured" the actual motion, although there are some errors at the boundaries of the volume (black regions), where there is lack of "texture". The estimated field is smoothly varying at the smooth volume regions, while it is not over-smoothed at object boundaries.

The corresponding estimation results of the LK method with two different integration windows ($6 \times 6 \times 6$ and $14 \times 14 \times 7$) are given in Figs. 12 and 13. The comparison with the proposed method is qualitatively difficult. The LK method also "captured" well the object's motion and presents only a few outliers. However, when using a small integration window, the estimated flow-field is not quite smooth and the method fails at some points in the textured object's interior. On the other hand, using the larger integration window, the motion field is over-smoothed at the object boundaries. In both cases, the LK method seems to produce smaller velocity estimates and does not "capture" well very large velocities, e.g. at the top-right of the object.

More details about the specific sequence and additional results can be found in section A.1 of the supplementary-material document.

6.3. Point-cloud sequence data

In this section, to assist inspection of the results, the directions of the motion vectors are encoded using a colormap, i.e. each motion vector is plotted with a color that depends on vector's direction in a (θ,ϕ) spherical representation. A HSV colormap of 8×8 distinct colors is constructed and used to encode the (θ,ϕ)-directions of the vectors. Additionally, in paragraph

Table 3: Armadillo1 sequence: Experimental comparative results and execution times.

Sequence: "Armadillo1" | Volume resolution: $65 \times 75 \times 63$		
Method	Overall MAE	Mean execution time
Proposed init-search (L=1)	11.21^o	8.09 sec
Proposed after optimization	8.56^o	17.31 sec
LK without	20.11^o	7.52 sec
LK with LP pre-filter	13.29^o	7.95 sec

6.3.2 the speed (magnitude of flow vectors) is encoded using a gray-scale colormap.

As explained in subsection 5.4 and will be verified, the computational time of the proposed algorithm in this section is lower than that in section 6.2, because the algorithm handles only the "occupied" voxels. The same holds for the LK algorithm, for the same reason.

6.3.1. Artificial data

For the needs of this work, a set of three artificial 3D point-cloud sequences, with known ground-truth (GT), was created. The well-known static 3D models "Armadillo" and "Dragon" of the Stanford 3D repository[2] were used and the sequences were generated using mesh Laplacian deformation [42]: For each sequence, a set of "handles" is defined on the static mesh, the "handles" are moved on a per-frame basis and the mesh is smoothly and realistically deformed. All sequences contain 21 frames and can be found at http://utopia.duth.gr/%7Enmitiano/f3sme/3dmotion%5Fdemo.html, along with the ground-truth (GT) motion. Additionally, details for the creation of each sequence can be found in the same URL address. In all experiments, the 3D flow algorithm is fed with $N_t = 5$ consecutive frames and the estimated flow is assigned to the middle frame. The integration window size of the proposed method is set to $11 \times 11 \times 11$voxels.

Sequence #1: "Armadillo1": In this sequence, the waist of the "Armadillo" creature does not move, while its upper-body and lower-body parts rotate clock-wise (CW) and counter-CW (CCW), respectively, with a constant rotational speed equal to from 3^o/frame. Using a voxel-size of 2mm, the described voxelization procedure generates volumes of $N_x \times N_y \times N_z =$

[2]Stanford 3D repository: http://graphics.stanford.edu/data/3Dscanrep/

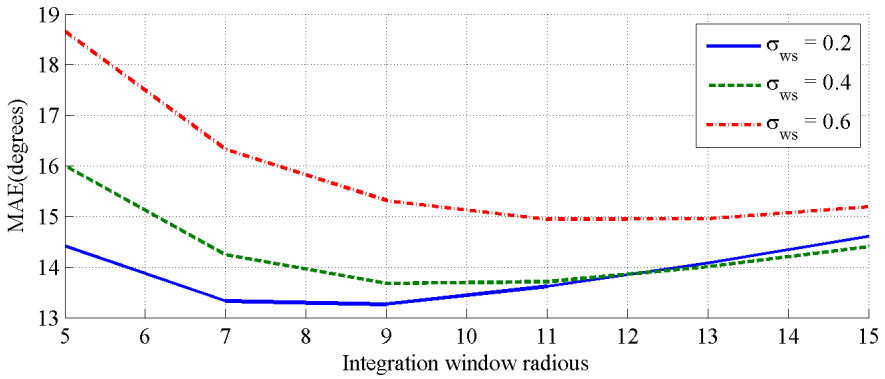


Figure 14: Sequence #1 ("Armadillo1"). Estimated MAE of the LK method, applied to the first $N_t = 5$ frames, for different integration windows and LP pre-filters.

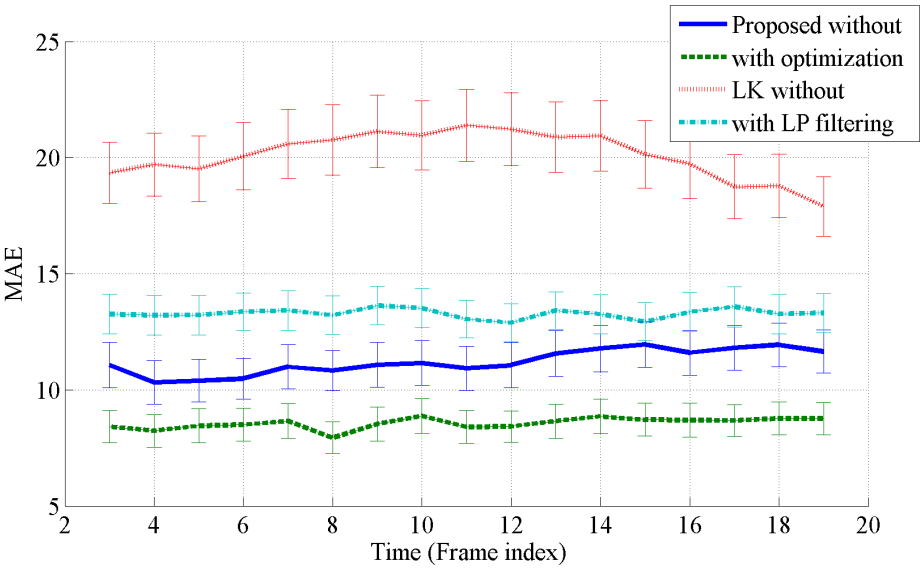
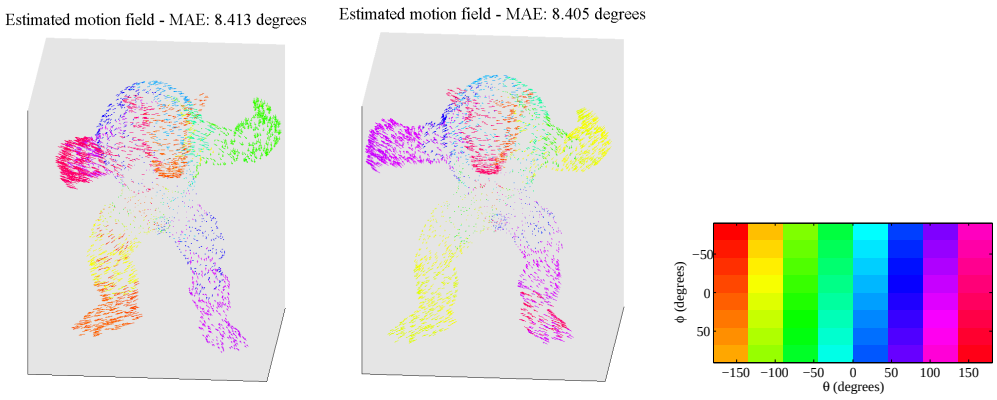


Figure 15: Sequence #1 ("Armadillo1"). Comparative MAE results long time.



(a) $t = 3$ (b) $t = 11$ (c)

Figure 16: Sequence #1: Experimental results for the "Armadillo1" sequence. Estimated 3D flow and MAE at (a) $t = 3$ and (b) $t = 11$; (c) Colormap used to depict the flows.

$65 \times 75 \times 63$ voxels. As stated, the algorithms were applied with $N_t = 5$ consecutive frames and the estimated flow is assigned to the central frame.

For the specific sequence, we experimented with simple directional filters of order $L = 1$. The velocity grid for the initial search is $7 \times 3 \times 7$. The 3D LK method was also executed without and with the application of a LP pre-filter, since the binary 0/1 input 3D signal contains high-frequency components. To select the integration window and the LP pre-filter, the LK method was applied to the first N_t=5 frames using various filters (σ_{ω_s} of the gaussian filter in the frequency domain) and various integration radii, as shown in the diagram of Fig.14. As can be seen, the optimal values are 9 voxels and σ_{ω_s}=0.2.

The MAE along time of the proposed method, before and after optimization, as well as of the LK method, with and without prefiltering, is given in Fig. 15. The corresponding overall MAE measures, along with the mean execution times, are summarized Table 3. One can observe that the proposed method produces quite low MAE in a reasonable execution time, outperforming the 3D LK method.

The estimated flow-fields of the proposed method after downhill simplex optimization are depicted for t=3 and t=11 in Figure 16, where the corresponding MAE measures are also given.

Sequence #2: "Armadillo2": In the specific sequence the creature's waist does not move, while its upper-body rotates about the horizontal (X) axis

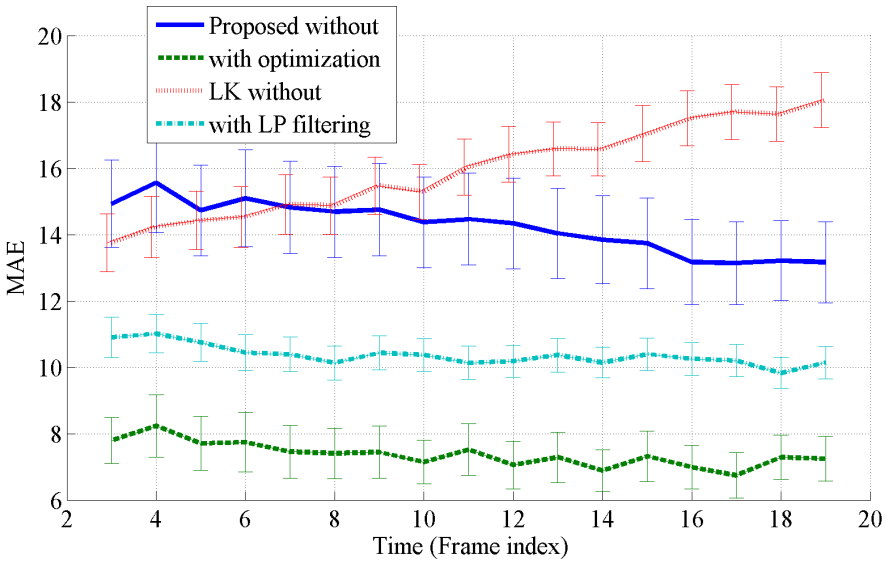
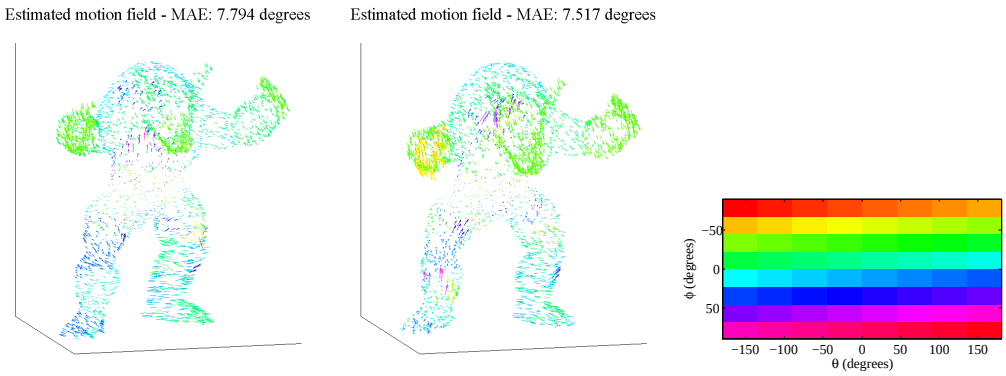


Figure 17: Sequence #2 ("Armadillo2"). Comparative MAE results long time.



(a) $t = 3$ (b) $t = 11$ (c)

Figure 18: Sequence #2: Experimental results for the "Armadillo2" sequence. Estimated 3D flow and MAE at (a) $t = 3$ and (b) $t = 11$; (c) Colormap used to depict the flows.

Table 4: Armadillo2 sequence: Experimental comparative results and execution times.

Sequence: "Armadillo2" | Volume resolution: $63 \times 75 \times 65$		
Method	Overall MAE	Mean execution time
Proposed init-search (L=2)	14.23^o	33.28 sec
Proposed after optimization	7.36^o	77.75 sec
LK without	15.94^o	7.41 sec
LK with LP pre-filter	10.35^o	8.01 sec

Table 5: Dragon sequence: Experimental results and execution times of the proposed method.

Sequence: "Dragon" | Volume resolution: $67 \times 49 \times 35$		
Method	Overall MAE	Mean execution time
Proposed init-search (L=1)	8.37^o	7.59 sec
Proposed after optimization	7.31^o	13.80 sec
LK without	10.41^o	5.90 sec
LK with LP pre-filter	8.14^o	6.25 sec

that pass through the center of gravity, from the erect position to a bent pose. On the other hand, its lower-body part rotates about the vertical (Y-)axis. The rotational speed for both the upper- and lower-body is constant, equal to 3^o/frame. The volume resolution for the specific sequence, using a voxel-size of 2mm, is $63 \times 75 \times 65$ voxels. The experimental results presented for this sequence were obtained using simple directional filters of order $L = 2$ and init search-grid $3 \times 5 \times 5$. The LK method was applied using the same parameters with the previous experiment, since the input 3D signal is the same (ignoring its motion).

As in the previous experiment, the MAE along time is given in Fig. 17, while the overall MAE measures and the mean execution times are summarized Table 4. In comparison to the 3D LK method, the proposed one produces smaller MAE errors. However, here, the execution-time of the proposed method is higher than in the previous experiment, due to the larger order of the employed filter.

The flow fields obtained by the proposed method after downhill optimization are given for two time instances (t=3 and t=11) in the diagrams of Figure 18.

Sequence #3: "Dragon": The dragon sequence contains quite compli-

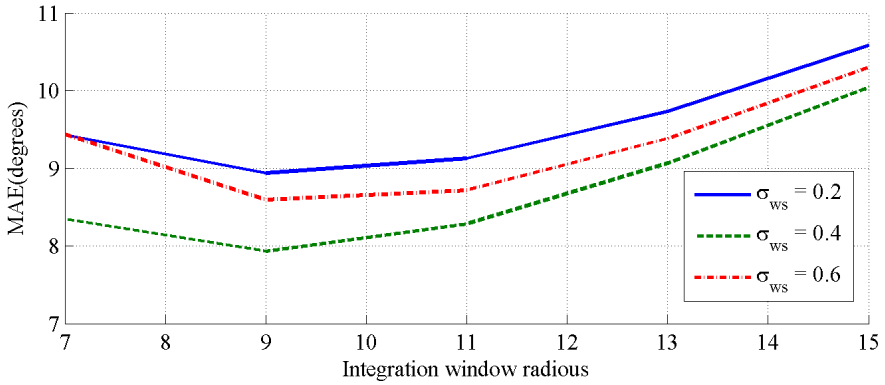


Figure 19: Sequence #3 ("Dragon"). LK method's estimated MAE for different integration windows and LP pre-filters.

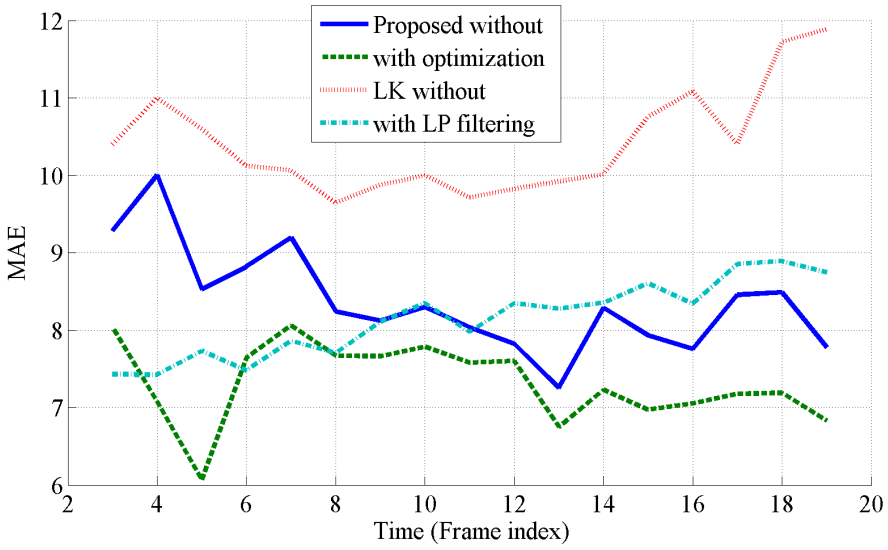


Figure 20: Sequence #3 ("Dragon"). Comparative MAE results long time.

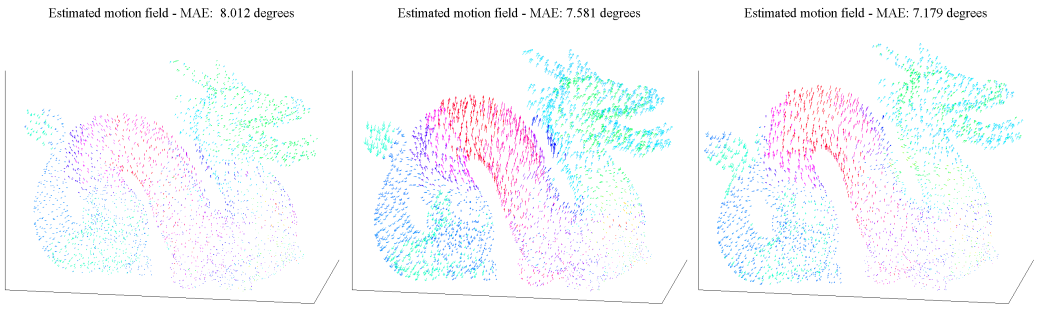


Figure 21: From Left to right $t = 3$, $t = 11$, $t = 17$. The used colormap is the same as the one in Fig. 18.

Figure 22: Sequence #3: Experimental results of the "Dragon" sequence. Estimated flow-field and MAE at $t = 3$, $t = 11$ and $t = 17$.

cated motions. More specifically, the bottom-frontal region of the "Dragon' creature deforms very slowly around a fixed "handle" point, while the position of the head varies sinusoidally with time and moves inwards. The middle part of the tail moves upwards with a sinusoid speed, whereas the whole tail deforms outwards.

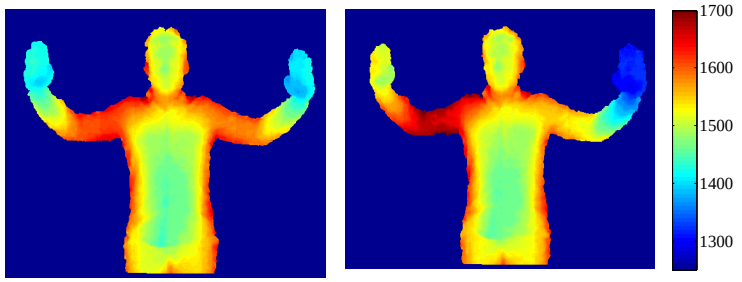
For the specific sequence, a voxel-size of 3mm is used, resulting into a volume resolution of $67 \times 49 \times 35$ voxels. For the results provided here, we used filters of order $L = 1$ and an init search grid $3 \times 9 \times 13$. The LK method was applied with an integration windows of radius equal to 9 voxels and a LP pre-filter of $\sigma_{\omega_s} = 0.4$, i.e. the optimal parameters found according to the diagrams of Fig. 19. The MAEs in the diagrams were obtained as the mean of the corresponding MAEs at $t = 6$, $t = 11$ and $t = 14$.

Comparative results along time are given in Fig. 20. The corresponding overall MAEs and mean execution times are given Table 5. In this experiment, although the proposed method in general outperforms the LK method, the latter produces quite accurate results, because the motion in voxels/frame is in general smaller than in the previous two experiments.

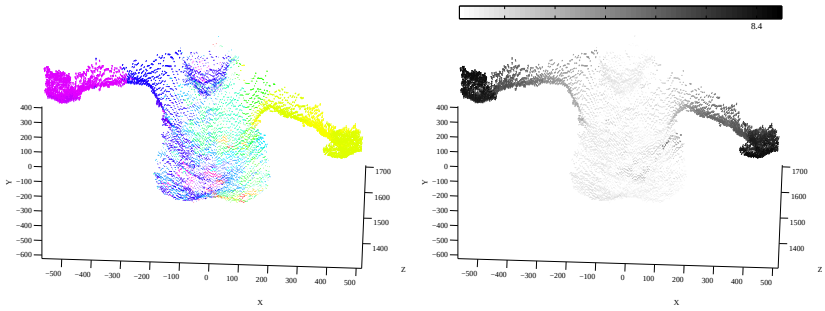
Example flow fields obtained by the proposed method after optimization, at t=3, 11 and 17, are given in Fig. 22.

6.3.2. Real-world Kinect data

In this subsection, we provide experimental results using sequences of real 3D point clouds, captured by one or multiple Microsoft Kinect sensors. More specifically, subsequences "Dimitris1" and "Dimitris2" constitute a part



(a) 1st and 16th frame (Depths)



(b) Estimated flow field

Figure 23: Sequence #4: (a) Kinect depth maps from which the "Dimitris1" sequence was generated; (b) Estimated flow-field. Flow vectors' direction is color encoded (left) using the colormap of Fig. 16(c). Speed is encoded with a gray-scale colormap (right).

of a long sequence, captured by a single Kinect and contain a human in various movements, while the "Dimitris Skiing" sequence was captured using five Kinects and contain a human moving fast on a ski simulator. Further details are given below. Since the GT of these sequences is not known, only qualitative conclusions can be drawn. In order to assist qualitative evaluation, apart from color encoding the directions of the motion vectors, we further encode the length of the vectors (speeds) using a gray-scale colormap. See for example Figure 23(b).

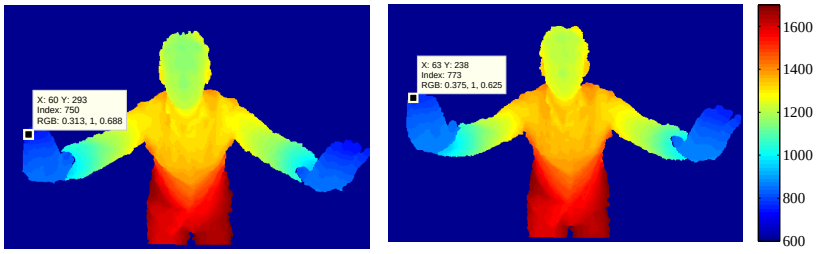
Sequence #4: "Dimitris1": This sequence contains a human, who's upper part rotates around his vertical body axis, clock-wise (CW). The motion is similar to upper-body motion of the artificial "Armadillo1". The motion is slow and therefore we experimented with N_t=16 volumes as input to the proposed algorithm. Kinect depth-maps, from which the 3D point clouds were reconstructed, are depicted in Figure 23(a). The human rotates around the vertical body axis, as can be observed.

In both "Dimitris1" and "Dimitris2" sequences, quadrature pairs of order $M = 2$ are employed. Additionally, for both sequences, a voxel size equal to 20mm is used. Notice that the point-cloud vertex positions corresponds to real-world units (milimeters) and therefore a size of 20mm (2cm) is a reasonable selection. For the specific voxel size, the volume resolution for "Dimitris1" is $58 \times 54 \times 28$ voxels.

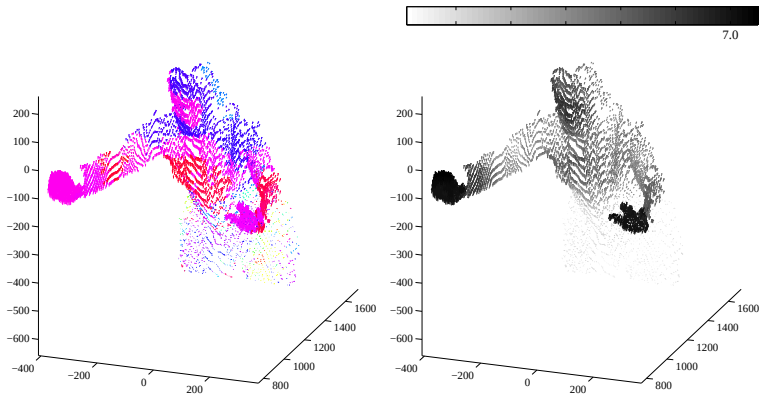
The computational times for this experiment are: 33.6sec for the initial grid search (a search-grid $7 \times 5 \times 7$ is used) and 15.4sec for downhill simplex optimization, resulting into totally 49sec. The obtained flow-field results are given in Figure 23(b). Qualitatively, the results are sensible, in accordance with the actual motion. For example, the opposite motion directions of the left/right body parts is reflected by the symmetric colors in the color-encoded diagram in Figure 23(b). Additionally, the more far a point is from the main vertical axis, the larger its speed is, as observed by the gray-scale encoded diagram in Figure 23(b).

Sequence #5: "Dimitris2": Depth-maps, from which the "Dimitris2" point-cloud sequence was reconstructed, are depicted Figure 24(a). The human holds his waist and lower-body almost static and rotates slowly his upper-part around the X-axis, from a bent pose towards an erect position. This results into a motion mainly upwards and towards back (far).

As with "Dimitris1", here the input of the proposed algorithm consists of N_t=16 volume frames. With the voxel-size of 20mm, the volume resolution



(a) 1st and 16th frame (Depths)



(b) Estimated flow field

Figure 24: Sequence #5: (a) Kinect depth maps from which the "Dimitris2" sequence was generated; (b) Estimated flow-field. Flow vectors' direction is color encoded (left) using the colormap of Fig. 18(c). Speed is encoded with a gray-scale colormap (right).

is $40 \times 48 \times 62$ voxels.

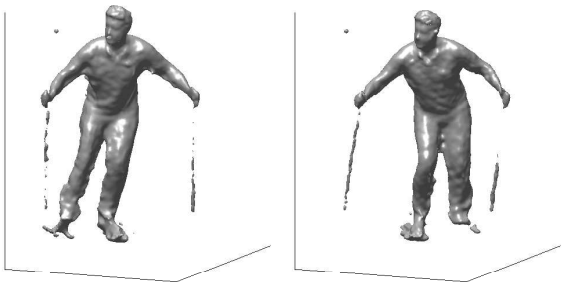
Using quadrature pairs of order M=2, the execution time for this experiment is totally 57.8sec, and more specifically 36.8sec for the initial grid search (a search-grid $5 \times 9 \times 5$ is used) and 21sec for downhill optimization. The flow-field results are depicted in Figure 24(b). The results are qualitatively correct, reflecting the actual motion. The more far a point from human's waist is, the larger the speed is. This is true in the right plot of Figure 24(b). Additionally, the dominant colors in the left plot of Figure 24(b) are magenta and red. According to the colormap in Figure 18(c), these colors encode the mainly upward motion of the human.

Sequence #6: "Dimitris Skiing3-PoissonLow": Finally, we present experimental results on a sequence of 360^o (full 3D) point-cloud data, reconstructed using depth maps from multiple Kinect sensors [43] by a watertight volumetric reconstruction method [38]. The data are freely available at http://vcl.iti.gr/reconstruction/ and constitute a part of the official MPEG-3DGC database [44].

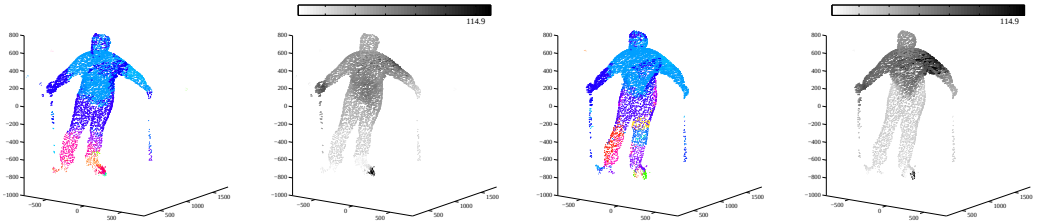
The sequence contains a human performing skiing on a ski simulator, as shown in Figure 25(a). The dominant motion is periodic and oscillatory, backwards-right, forwards-left. The motion is quite fast. Notice from Figure 25(a) that a single half-period (backwards-right) is less than 6 frames. Therefore, here we use as input to the algorithm N_t=4 frames. Additionally, the used voxel size is 35mm, resulting into volumes of approximately $40 \times 56 \times 48$ voxels (this changes slightly, as the algorithm is fed with consecutive frame quartets).

We experimented with simple directional filter of order $L = 1$. The output of the proposed algorithms on 60 frames of the sequence (frames 240-299) is given in the supplementary-material video. The mean execution time from the 60 runs of the algorithm is 6.7sec/frame, 4sec for the initial grid search (a search-grid $7 \times 5 \times 5$ is used) and 2.7sec for downhill optimization.

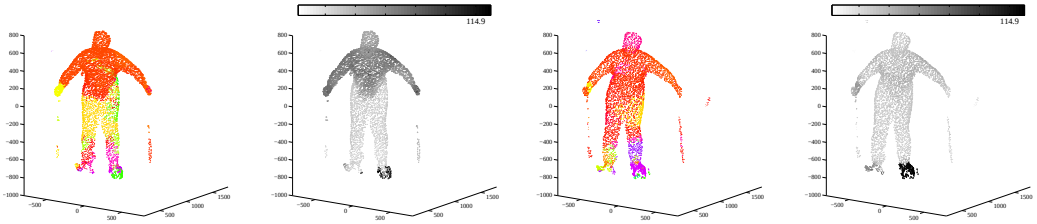
In Figures 25(b) and (c), we provide the experimental results for two phases of the skiing performance. More specifically, Figure 25(b) depicts the estimated fields at the beginning of a backwards-right phase, while Figure 25(c) shows the fields at the end of a forwards-left phase. One can observe in Figure 25(b) the domination of cold colors (cyan and blue), reflecting the backwards-right motion. Even the presence of red at the leg of the skier is qualitatively correct, since the leg moves in the opposite direction, due to motion inertia. Additionally, the fact that the estimated speeds are initially



(a) Skiing Frames 257 and 261

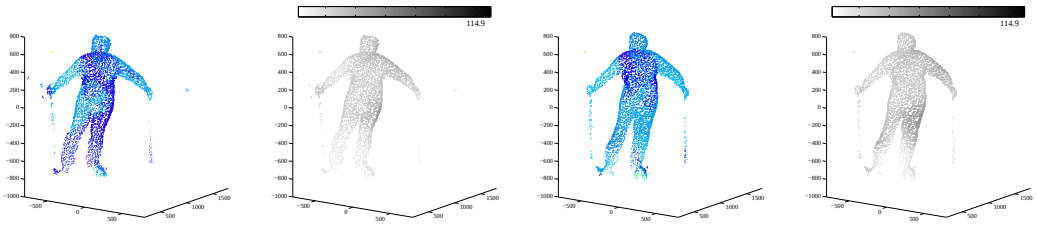


(b) Skiing Part 1: Estimated flow-fields for t = 256, 257

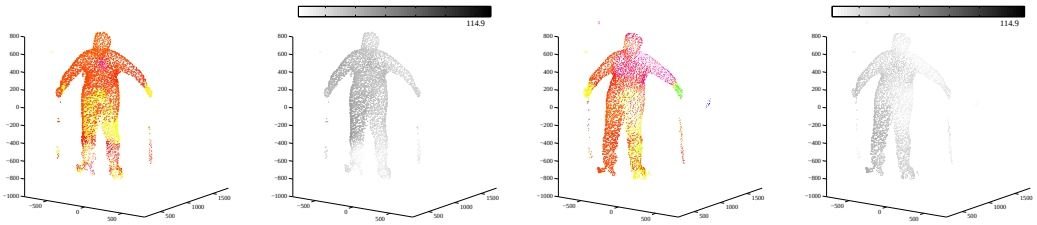


(c) Skiing Part 2: Estimated flow-fields for t = 265, 266

Figure 25: Sequence #6 "Skiing": (a) Two Frames of the sequence (apprx. one half-period); (b),(c) Estimated flow-fields.



(a) Skiing Part 1: Estimated flow-fields for t = 256, 257



(b) Skiing Part 2: Estimated flow-fields for t = 265, 266

Figure 26: Sequence #6 "Skiing": (a),(b) Estimated flow-fields of the LK method (7×7×7) and $\sigma_{\boldsymbol{\omega}_s}$=0.2.

small and increase as the human accelerates, are in accordance with the actual motion. On the other hand, in Figure 25(c) hot colors dominate (red and magenta), reflecting the forwards-left motion. Additionally, as can be seen at the right part of the figure, the speeds are initially large and decrease as the human decelerates his motion, before stoping. The estimated fields, qualitatively are in a accordance with the actual motion of the skier.

In Fig.26, the corresponding results of the LK method are given. The method was applied with a LP-prefilter of $\sigma_{\boldsymbol{\omega}_s}$=0.2, as in the "Armandillo" squences, and with an integration window of size $7 \times 7 \times 7$. This window size is in relation with the window used in the "Armandillo" sequences, since their volume size was approximately $1\frac{1}{2}$ times larger along each dimension. Still, qualitative comparison is difficult. One observation is that the LK method outputs smaller velocities (lighter gray in the images) and it seems that it cannot capture well the fast motion of the upper body part.

Apart from the supplementary video, detailed experimental results for the specific sequence can be found in subsection A.2 of the supplementary-material document.

7. Discussion - Conclusions

In this work, the theoretical details for the construction of narrow multi-dimensional steerable filters and quadrature pairs were given. Although the constructed filters and quadrature pairs can serve several multi-dimensional signal processing tasks, we focused on the dense 3D flow estimation problem directly in three dimensions. By studying the problem in the N-D spatiotemporal frequency domain we showed that motion manifests itself as a "motion hyper-plane" in the frequency domain. Guided by this fact, we formulated the "Hyper-donut" mechanism for both simple directional filters and quadrature pairs and provided a thorough theoretical analysis on how it can be employed for 3D flow estimation. Finally, based on our theoretical developments, a simple yet efficient algorithm for 3D flow estimation in volume or point-cloud sequences was described.

The proposed algorithm, based on the introduced steerable filters, present low computational effort when low-order filters are used (L=1), despite the high dimensionality of the data. This effort is comparable with that of spatiotemporal differential approaches, such as the extension of Lucas-Kanade (LK) into 3 dimensions. However, as the used filters' order increase the computational effort increases, producing potentially more accurate results. We note that the given and tested algorithm is the simplest one that could be formulated based on our theoretical developments. Compared to differential approaches, such as the LK, the algorithm is more accurate, according to the presented experimental results, regardless of the used filters' order. Similar results are expected with other differential approaches, such as with the Horn-Schunck (HS) method, which is less accurate than LK, according to [13]. Additionally, given the high dimensionality of the data, a global approach (i.e. estimation of the flow by minimizing a global function for the whole volume), such as the HS method, is expected to present high computational and memory requirements.

One limitation of the proposed approach, as analyzed in subsection 5.4, is its high memory requirements. Thus, given the current technology, it cannot handle very large volume data, unless a sophisticated implementation is investigated.

Apart from analyzing volume data sequences (as e.g. in [16, 17, 18, 19]), possible applications of the presented approach include analysis of 3D point-cloud/meshes, for instance for 3D scene analysis (e.g. human action recognition tasks [45]) or in time-varying mesh compression [46], to align a source

point-cloud to a target one.

The directions of future work include a) the implementation and experimentation with alternative, more sophisticated algorithmic ideas (e.g. see subsection 5.2) and b) the parallel-computing implementation of the algorithm on the GPU (Graphical Processing Unit), e.g. using NVidia's CUDA (Compute Unified Device Architecture). The described algorithm involve mainly voxelwise operations that can be parallelized in several execution threads. As demonstrated in our previous work [10], a GPU implementation can introduce a significant speed-up of the algorithm. Finally, c) one of our intentions is to test the proposed 3D flow estimation framework in human motion/action analysis and recognition tasks, as e.g. in [45].

Acknowledgement

This work was supported by the F3SME research project (PE6 3210), implemented within the framework of the Action Supporting Postdoctoral Researchers of the Operational Program Education and Lifelong Learning, and co-financed by the European Social Fund (ESF) and the Greek State.

Appendix A. Supplementary theoretical notes

In this Appendix, some additional notes are provided to support the theoretical developments of the paper.

Appendix A.1. Hyper-spherical coordinates and integrals

Appendix A.1.1. Hyper-spherical coordinates

The hyper-spherical coordinates in N dimensions are defined by the radius r and $N-1$ spherical angles $\{\phi_n\}_{n=0,\ldots,N-2}$, with $\phi_0 := \theta$ ranging in $[0, 2\pi)$ radians and $\{\phi_n\}_{n>0}$ ranging in $[0, \pi)$ radians. The hyper-spherical to cartesian transformation is recursively defined, as follows [29, ch. 22]:

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \Psi_n(r\ ;\ \theta, \phi_1, \ldots, \phi_{n-2}) = \begin{pmatrix} \Psi_{n-1}\big(r\sin(\phi_{n-2})\ ;\ \theta, \phi_1, \ldots, \phi_{n-3}\big) \\ r\cos(\phi_{n-2}) \end{pmatrix}, \tag{A.1}$$

where $\Psi_2(r\ ;\ \theta) = \begin{pmatrix} r\cos\theta \\ r\sin\theta \end{pmatrix}$ the well-known polar-to-cartesian transformation.

Appendix A.1.2. Hyper-spherical integrals

We consider a function defined on the unit hyper-sphere[c] S^{N-1}, denoted as $g(\boldsymbol{\phi})$, where $\boldsymbol{\phi} = (\theta, \phi_1, \ldots, \phi_{N-2}) \in [0, 2\pi) \times [0, \pi)^{N-2}$. The surface integral is given [29, ch. 22] (with $r = 1$) by:

$$\int_{\boldsymbol{\phi}} g(\boldsymbol{\phi}) \;\; \Delta_N(\theta, \phi_1, \ldots, \phi_{N-2}) \;\; d\theta \; d\phi_1 \; d\phi_2 \ldots d\phi_{N-2}, \tag{A.2}$$

where the Jacobian is $\Delta_N(\theta, \phi_1, \ldots, \phi_{N-2}) = \sin\phi_1 \;\; \sin^2\phi_2 \ldots \sin^{N-2}(\phi_{N-2})$. For example, with $g(\boldsymbol{\phi}) = 1$ the surface of the hyper-sphere is

$$\begin{aligned} \sigma(S^{N-1}) &= \int_{\boldsymbol{\phi}} \Delta_N(\theta, \phi_1, \ldots, \phi_{N-2}) \;\; d\theta \; d\phi_1 \; d\phi_2 \ldots d\phi_{N-2} \\ &= 2\pi \prod_{n=1}^{N-2} \gamma_n, \quad \text{where } \gamma_n = \int_{\phi=0}^{\pi} \sin^n\phi \;\; d\phi. \end{aligned} \tag{A.3}$$

For the unit-circle it is $\sigma(S^1) = 2\pi$, for the unit-sphere $\sigma(S^2) = 2 \cdot 2\pi$, etc.

For the needs of the paper (see subsection 3.2.1), we consider a function that depends only on the last spherical angle ϕ_{N-2}, i.e. $g(\phi_{N-2})$. Based on the previous analysis, the surface integral on a portion of the unit hyper-sphere $\phi_{N-2} \in [\Phi_1, \Phi_2]$ is

$$E(\Phi_1, \Phi_2) = \sigma(S^{N-2}) \cdot \int_{\Phi_1}^{\Phi_2} g(\phi_{N-2}) \;\; \sin^{N-2}(\phi_{N-2}) \;\; d\phi_{N-2}. \tag{A.4}$$

Appendix A.2. Calculation of trigonometric power integrals

For the needs of subsection 3.2.1, the calculation of the integral

$$R(m) = \int \cos^m\phi \;\; \sin^{N-2}\phi \;\; d\phi \tag{A.5}$$

is needed, where $m = k + l$. Considering the 4D case ($N = 4$), here we derive an analytical solution of this integral. With $N = 4$, it holds:

$$\begin{aligned} R(m) &= \int \cos^m\phi \;\; \sin^2\phi \;\; d\phi = \int \cos^m\phi \left(1 - \cos^2\phi\right) \;\; d\phi \\ &= \int \cos^m\phi \, d\phi - \int \cos^{m+2}\phi \;\; d\phi = P(m) - P(m+2), \end{aligned} \tag{A.6}$$

[c] To be specific, S^1 is the unit circle, S^2 the unit sphere, etc.

where $P(n) = \int \cos^n \phi \;\; d\phi$.

Using Euler's formula $e^{\mathtt{j}\phi} = \cos\phi + \mathtt{j}\sin\phi$, the binomial expansion theorem and De Moivre's formula $(\cos\phi + \mathtt{j}\sin\phi)^n = \cos(n\phi) + \mathtt{j}\sin(n\phi)$, after a set of manipulations and making use of trigonometric identities, one can conclude to the "cosine-power reduction" formula:

$$\cos^n \phi = D_n + \frac{1}{2^{n-1}} \sum_{k=0}^{\llcorner \frac{n-1}{2} \lrcorner} \left(\begin{array}{c} n \\ k \end{array} \right) \cos\big((n-2k)\phi\big), \tag{A.7}$$

where $\llcorner \circ \lrcorner$ stands for the floor (integer part) operation and

$$D_n = \begin{cases} 0 & \text{if } n \text{ odd} \\ \frac{1}{2^n} \left(\begin{array}{c} n \\ n/2 \end{array} \right) & \text{if } n \text{ even.} \end{cases} \tag{A.8}$$

Based on the above formula, one concludes to

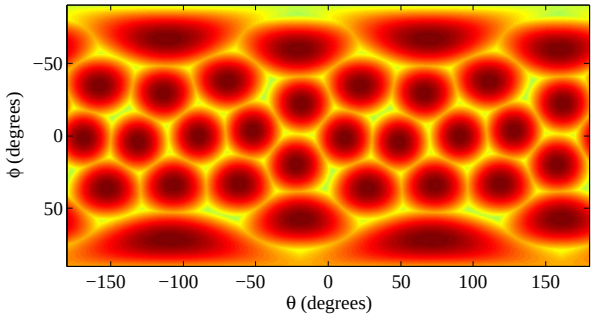
$$P(n) = \int \cos^n \phi \;\; d\phi = C_n + \frac{1}{2^{n-1}} \sum_{k=0}^{\llcorner \frac{n-1}{2} \lrcorner} \left(\begin{array}{c} n \\ k \end{array} \right) \frac{\sin\big((n-2k)\phi\big)}{n-2k}, \tag{A.9}$$

where C_n is zero for odd n and $C_n = \frac{1}{2^n} \left(\begin{array}{c} n \\ n/2 \end{array} \right) \phi$, otherwise.

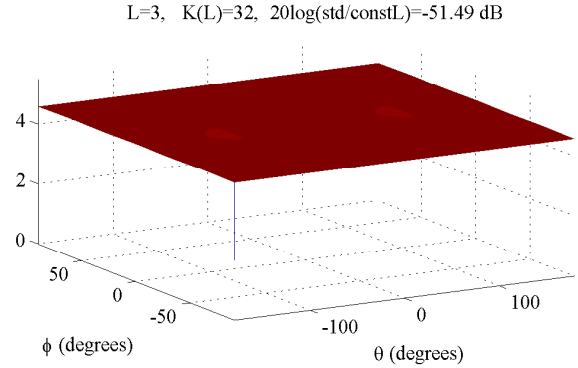
Appendix A.3. Extended cross-product for N dimensions

The cross-product is defined in N dimensions as the operation that takes $N-1$ vectors and produces a new one which is perpendicular to them. Let the standard basis of $\mathbb{R}^N$ be $\{\mathbf{i}_1, \mathbf{i}_2, \ldots, \mathbf{i}_N\}$. Let also $\times\Big(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{N-1}\Big)$ denote the extended cross-product of $N-1$ vectors in N dimensions, where $\mathbf{w}_i = [w_{i,1}, w_{i,2}, \ldots, w_{i,N}]^{\mathtt{T}}$. The extended cross-product is calculated from [30]:

$$\times\Big(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{N-1}\Big) = \begin{vmatrix} \mathbf{i}_1 & \mathbf{i}_2 & \ldots & \mathbf{i}_N \\ w_{1,1} & w_{1,2} & \ldots & w_{1,N} \\ \vdots & \vdots & \vdots & \vdots \\ w_{N-1,1} & w_{N-1,2} & \ldots & w_{N-1,N} \end{vmatrix}. \tag{A.10}$$



(a) $\max_k |B_k(\theta,\phi)|, \quad k = 1,\ldots,K(L)$



(b) $\sum_k \left(B_k(\theta,\phi)\right)^2$

Figure B.27: Pictorial demonstration of eq. (B.1).

Appendix B. Demonstration of the "Hyper-donut" mechanism in the 4D case

Appendix B.1. Demonstration of equation (21)

We would like to demonstrate the validity of equation (21) in the 4D case. Without loss of generality, let $\mathbf{v} = \mathbf{0}^\mathsf{T}$ and therefore, according to (18), it is $\mathbf{n}(\mathbf{v}) = [\mathbf{0}^\mathsf{T}, 1]^\mathsf{T}$. From equation (19) with $\mathbf{v} = \mathbf{0}^\mathsf{T}$, the vectors spanning the "motion hyper-plane" are $\mathbf{e}_n(\mathbf{v}) = \mathbf{i}_n(\mathbf{v}), \quad n = 1, 2, \ldots, N-1$, namely they match the basis of the (N-1)-D subspace. Consequently, from (20), we have

$$\mathbf{v} = \mathbf{0}^\mathsf{T} \Rightarrow \mathbf{s}_k(\mathbf{v}) = \mathbf{q}_k, \quad k = 1, 2, \ldots, K.$$

Therefore, it remains to verify that

$$\textit{On motion hyper-plane } (\forall \boldsymbol{\omega} \perp \mathbf{n}(\mathbf{v})): \quad \sum_{k=1}^{K(L)} \left(\hat{\boldsymbol{\omega}}^\mathsf{T} \cdot \mathbf{q}_k\right)^{2L} = \text{const}(L) \quad \text{(B.1)}$$

holds on the "motion hyper-plane" that is perpendicular to $\mathbf{n}(\mathbf{v}) = [\mathbf{0}^\mathsf{T}, 1]^\mathsf{T}$.

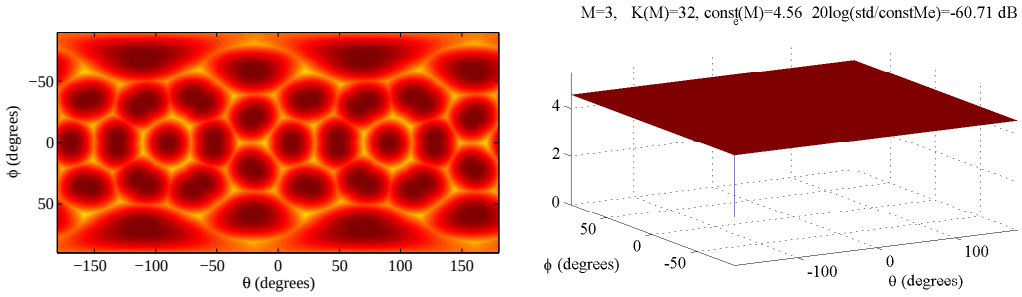
The hyper-plane that is perpendicular to $\mathbf{n}(\mathbf{v}) = [\mathbf{0}^\mathsf{T}, 1]^\mathsf{T}$, is $\omega_4 = 0$. Therefore, the unit-frequency vector on this hyper-plane is of the form $\hat{\boldsymbol{\omega}} = [\hat{\boldsymbol{\omega}}_{3\text{D}}^\mathsf{T}, 0]^\mathsf{T}$, where $\hat{\boldsymbol{\omega}}_{3\text{D}} = [\hat{\omega}_1, \hat{\omega}_2, \hat{\omega}_3]^\mathsf{T}$ is a 3D unit-frequency vector. In other words, $\hat{\boldsymbol{\omega}}_{3\text{D}}$ lies on the unit sphere. Consequently, exploiting spherical coordinates, we use the term $B_k(\theta, \phi)$ to notate the underlying directional filters that are equally distributed on the unit sphere:

$$B_k(\theta, \phi) \longleftarrow B_k(\hat{\boldsymbol{\omega}}) := (\hat{\boldsymbol{\omega}}^\mathsf{T} \cdot \mathbf{q}_k)^L, \quad k = 1, \ldots, K \quad \text{(B.2)}$$

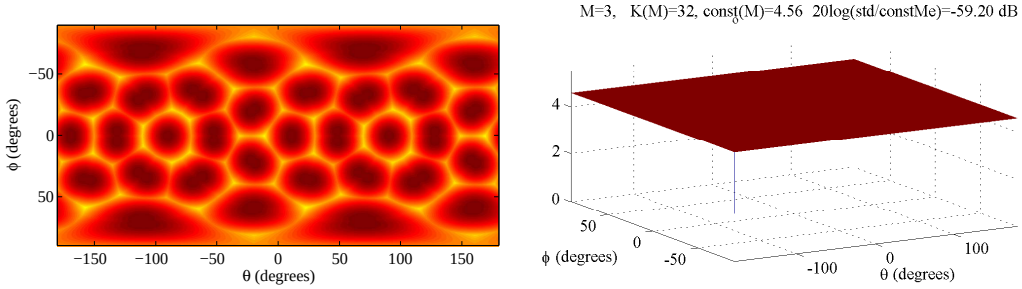
where (θ, ϕ) is the spherical-coordinates representation of the unit vector $\hat{\boldsymbol{\omega}}_{3\text{D}}$.

In other words, notice here that by definition of the filters introduced in this paper, a filter's value (in the frequency domain) is independent to $||\boldsymbol{\omega}||$. It depends only on $\hat{\boldsymbol{\omega}} = \boldsymbol{\omega}/||\boldsymbol{\omega}||$, i.e. on the filter's direction. Therefore, for the studied case of $\mathbf{v} = \mathbf{0}^\mathsf{T}$, one can depict the filters' values as a function of orientation (θ, ϕ).

Demonstrations in Fig. B.27 are given for $L = 3$ and consequently $K(L) = 32$. Figure B.27(a) depicts the filters $|B_k(\theta, \phi)|, k = 1, \ldots, 32$. To be more specific, it depicts $\max_{k=1}^{K} |B_k(\theta, \phi)|$. Given that the vectors $\mathbf{q}_k$ are equally distributed on the unit sphere, the uniform distribution of the filters' directions is evident in Fig. B.27(a).



(a) Left: $\max_k |Q_k^{\text{even}}(\theta,\phi)|$, Right: $\sum_k \left(Q_k^{\text{even}}(\theta,\phi)\right)^2$



(b) Left: $\max_k |Q_k^{\text{odd}}(\theta,\phi)|$, Right: $\sum_k \left(Q_k^{\text{odd}}(\theta,\phi)\right)^2$

Figure B.28: Pictorial demonstration of eq. (27).

Based on the above, in order to demonstrate (B.1), it is adequate to show that the energy $\sum_{k=1}^{K} \left(B_k(\theta,\phi)\right)^2$ is constant, independent to (θ,ϕ). This fact is depicted in Figure B.27(b).

Appendix B.2. Demonstration of equation (27) *- Quadrature pairs*

Using similar notation and arguments with the previous subsection, the validity of equation (27), i.e. the "Hyper-donut" mechanism for quadrature pairs in the 4D case, is demonstrated here for $M = 4$ and $K(M) = 32$.

The following notation is used

$$Q_k^{\text{even}}(\theta,\phi) := Q_k^{\text{even}}(\hat{\boldsymbol{\omega}}) := \sum_{L=0,L+=2}^{M} a_M[L](\hat{\boldsymbol{\omega}}^{\mathrm{T}} \cdot \mathbf{q}_k)^L,$$

$$Q_k^{\text{odd}}(\theta,\phi) := Q_k^{\text{odd}}(\hat{\boldsymbol{\omega}}) := \sum_{L=1,L+=2}^{M} a_M[L](\hat{\boldsymbol{\omega}}^{\mathrm{T}} \cdot \mathbf{q}_k)^L. \tag{B.3}$$

Figure B.28(a) depicts $|Q_k^{\text{even}}(\theta,\phi)|, k = 1,\ldots,32$. It additionally shows the energy $\sum_k \left(Q_k^{\text{even}}(\theta,\phi)\right)^2$, which is almost constant, independent to (θ,ϕ) and equal to $\text{const}_{\text{e}}(M) = 4.56$. Similarly, Figure B.28(b) depicts the corresponding results for the odd part of the quadrature filters. It can be verified that $|Q_k^{\text{odd}}(\theta,\phi)| \simeq |Q_k^{\text{even}}(\theta,\phi)|$, as well as $\text{const}_{\text{o}}(M) = \text{const}_{\text{e}}(M) = 4.56$.

References

[1] W. T. F. Freeman, E. H. Adelson, The design and use of steerable filters, IEEE Trans. Pattern Anal. Mach. Intell. 13 (9) (1991) 891 – 906.

[2] W. T. Freeman, Steerable filters and local analysis of image structure, Ph.D. thesis, School of Architecture and Planning, Massachusetts Institute of Technology (1992).

[3] D. S. Alexiadis, G. D. Sergiadis, Narrow directional steerable filters in motion estimation, Computer Vision and Image Understanding 110(2) (2008) 192–211.

[4] E. H. Adelson, J. R. Bergen, Spatiotemporal energy models for the perception of motion, J. Opt. Soc. Am. A 2 (Feb.1985) 284–299.

[5] E. P. Simoncelli, Distributed representation and analysis of visual motion, Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (1993).

[6] S. Vedula, S. Baker, P. Rander, R. Collins, T. Kanade, Three-dimensional scene flow, IEEE Trans. Pattern Anal. Mach. Intell. 27 (Mar. 2005) 475–480.

[7] C.-L. Huang, Y.-T. Chen, Motion estimation method using a 3D steerable filter, Image and Vision Computing 13 (1) (1995) 21–32.

[8] K. Derpanis, J. M. Gryn, Three-dimensional n-th derivative of Gaussian separable steerable filters, in: IEEE International Conference on Image Processing (ICIP), 2005.

[9] M. T. Andersson, Controllable multi-dimensional filters and models in low-level computer vision, Ph.D. thesis, Department of Electrical Engineering, Linkonping University, Sweden (1992).

[10] D. Alexiadis, N. Mitianoudis, T. Stathaki, Multidimensional steerable filters and 3D flow estimation, in: IEEE International Conference on Image Processing (ICIP), 2014, pp. 2012 –2016.

[11] B. Horn, B. Schunck, Determining optical flow, Artif. Intel. 17 (1981) 185–204.

[12] B. Lucas, T. Kanade, An iterative image registration technique with an application to stereo vision, in: Proc. Seventh International Joint Conference on Artificial Intelligence, 1981, pp. 674–679.

[13] J. Barron, D. Fleet, S. Beauchemin, Performance of optical flow techniques, Int. J. Comput. Vision 12(1) (1994) 43–77.

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

[15] J. L. Barron, N. A. Thacker, Tutorial: Computing 2D and 3D Optical Flow, Tina Memo No. 2004-012, 2005.

[16] J. L. Barron, Experience with 3D optical flow on gated MRI cardiac datasets, in: 1st Canadian Conference on Computer and Robot Vision, 2004, pp. 370–377.

[17] M. D. Abramoff, M. A. Viergever, Computation and visualization of three-dimensional soft tissue motion in the orbit, IEEE Transactions on Medical Imaging 21 (4) (2002) 296–304.

[18] X. Chen, J. L. Barron, R. E. Mercer, P. Joe, 3D regularized velocity from 3D doppler radial velocity, in: IEEE Int. Conf. Image Processing (ICIP), Vol. 3, Thessalonki, Greece, 2001, pp. 664–667.

[19] J. L. Barron, R. E. Mercer, X. Chen, P. Joe, 3D velocity from 3D doppler radial velocity, Int. J. of Imaging Systems and Technology 15 (3) (2005) 189–198.

[20] A. A. Kassim, P. Yan, W. S. Lee, K. Sengupta, Motion compensated lossy-to-lossless compression of 4-D medical images using integer wavelet transforms, IEEE Trans. on Information Technology in Biomedicine 9 (Mar. 2005) 132–138.

[21] L. Alvarez, C. A. Castano, M. Garcia, K. Krissian, L. Mazorra, A. Salgado, J. Sanchez, 3D motion estimation using a combination of correlation and variational methods for PIV, in: EUROCAST 2007, LNCS 4739, 2007, pp. 612–620.

[22] H. Spies, B. Jaehne, J. L. Barron, Range flow estimation, Computer Vision and Image Understanding 85 (2002) 209–231.

[23] M. B. Holte, B. Chakraborty, T. B. Moeslund, J. Gonzalez, A local 3D motion descriptor for multi-view human action recognition from 4d spatio-temporal interest points, IEEE Journal of Selected Topics in Signal Processing, Special Issue on emerging techniques in 3D 6 (Sep. 2012) 553 – 565.

[24] H. Li, R. W. Sumner, M. Pauly, Global correspondence optimization for non-rigid registration of depth scans, Computer Graphics Forum, Proc. of the Sixth Eurographics Symposium on Geometry Processing, 2008 27 (5).

[25] H. Li, B. Adams, L. J. Guibas, M. Pauly, Robust single-view geometry and motion reconstruction, ACM Transactions on Graphics.

[26] M. Hazewinkel (Ed.), Multinomial coefficient, Encyclopedia of Mathematics, Springer, 2001.

[27] E. B. Saff, A. B. Kuijlaars, Distributing many points on a sphere, The Mathematical Intelligencer 19 (1) (1997) 5–11.

[28] B. Jaehne, Digital Image Processing, 6th revised and extended Edition, Springer, 2005.

[29] B. K. Driver, Analysis Tools with Applications, Springer, 2003.

[30] M. Spivak, Calculus on Manifolds: A Modern Approach To Classical Theorems Of Advanced Calculus, 1971.

[31] M. Frigo, S. G. Johnson, The design and implementation of FFTW3, Proceedings of the IEEE 93 2 (2005) 216–231.

[32] S. G. Johnson, M. Frigo, A modified split-radix FFT with fewer arithmetic operations, IEEE Trans. Signal Processing 55 (1) (2007) 111–119.

[33] W. Press, S. A. Teukolsky, W. Vetterling, B. Flannery, Numerical Recipes: The Art of Scientific Computing, 3rd Edition, New York: Cambridge University Press, 2007.

[34] O. Cappe, S. J. Godsill, E. Moulines, An overview of existing methods and recent advances in sequential monte carlo, Proceedings of the IEEE 95 (5).

[35] J. Gall, J. Potthoff, C. Schnoerr, B. Rosenhahn, H.-P. Seidel, Interacting and Annealing Particle Filters: Mathematics and a recipe for applications, Journal of Mathematical Imaging and Vision 28 (1) (May 2007) 1–18.

[36] K. He, J. Sun, X. Tang, Guided image filtering, IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (6) (2013) 1397–1409.

[37] D. Alexiadis, D. Zarpalas, P. Daras, Real-time, realistic full-body 3D reconstruction and texture mapping from multiple kinects, in: 11th IEEE IVMSP Workshop, June 2013.

[38] M. Kazhdan, M. Bolitho, H. Hoppe, Poisson surface reconstruction, in: Symp. on Geometry Processing, 2006, pp. 61–70.

[39] E. P. Simoncelli, Design of multi-dimensional derivative filters, in: IEEE Int. Conf. Image Processing, Vol. 1, 1994, pp. 790–793.

[40] Organ motion from 4D MRI, Online.
URL `www.vision.ethz.ch/4dmri`

[41] M. von Siebenthal, G. Szekely, U. Gamper, P. Boesiger, A. Lomax, P. Cattin, 4D MR imaging of respiratory organ motion and its variability, Phys. Med. Biol. 52 (2007) 1547–1564.

[42] Y. Lipman, O. Sorkine, D. Cohen-Or, D. Levin, C. Roessl, H.-P. Seidel, Differential coordinates for interactive mesh editing, in: Shape Modeling Applications, 2004.

[43] D. Alexiadis, D. Zarpalas, P. Daras, Real-time, full 3-D reconstruction of moving foreground objects from multiple consumer depth cameras, IEEE Transactions on Multimedia 15(2) (Feb. 2013) 339–358.

[44] MPEG-3DGC database, Online.
URL `http://www.gti.ssr.upm.es./%7Empeg/3dgc/`

[45] G. T. Papadopoulos, P. Daras, Human action recognition using 3d reconstruction data, IEEE Trans. on Circuits and Systems for Video Technology.

[46] A. Doumanoglou, D. Alexiadis, D. Zarpalas, P. Daras, Towards real-time and efficient compression of human time-varying-meshes, IEEE Trans. on Circuits and Systems for Video Technology (24).