FLUID Image analysis and Description (FLUID)
Project No. FP6-513663
FET - Open Domain, Priority IST

Deliverable 4.2

3D motion estimation of fluid flows

(1)L. Alvarez, C.A. Castaño, M. García, K. Krissian, L. Mazorra, A. Salgado, J. Sánchez

(1)AMI group, Univ. Las Palmas Gran Canaria

Due date of deliverable: 30 June 2007
Actual submission date: 30 July 2007
Start date of project: 1 December 2004
Duration: 3 years

Project cofunded by the European Comission
within the Sixth Framework Programme (2002-2006).

Dissemination level: PU (Public)  Revision: 2

1lalvarez@dis.ulpgc.es
Contents

1 Introduction 3

2 Extension of 2D basic motion estimation techniques to 3D 3
  2.1 Cross-Correlation ................................................. 3
    2.1.1 Basic Principle ............................................. 4
    2.1.2 Implementation using Fast Fourier Transform .......... 4
  2.2 Variational Approach ........................................... 5
    2.2.1 Numerical Scheme ........................................... 5

3 Helmholtz vector field decomposition 6

4 Incompressibility constrained energy model 7

5 Nonlinear discrete optimisation problem. 8

6 Experiments and Results on Synthetic images 10
  6.1 Choice of the parameters ........................................ 10
  6.2 Description of the models ...................................... 10
  6.3 Experiments with model 1 (sphere) ........................... 12
  6.4 Experiments with model 2 (cylinder) ......................... 13
  6.5 Experiments with model 3 (cylinder with low particle density) . 14

7 Experiments and Results on LaVision Real Image dataset 17

8 Software for 3D motion estimation 21
  8.1 C/C++ Interface .................................................. 21
  8.2 AMILab Interface .................................................. 21
  8.3 ami_optic_flow_correlation_3d .................................. 22
  8.4 ami_optic_flow_correlation_3d_new ............................ 22
  8.5 ami_horn_schunck_optic_flow_3d ................................ 23
  8.6 PDEOpticFlowParam .............................................. 23
  8.7 amiOFCorr3D ...................................................... 24
  8.8 amiOFCorr3DNew .................................................. 24
  8.9 amiOFPDE .......................................................... 24
  8.10 3D Warping ....................................................... 25
  8.11 3D flow estimation with the incompressibility constraint .... 25
  8.12 Solenoidal projection ............................................ 25
  8.13 Example of Graphical Interface in AMILab ................... 25
1 Introduction

Particle Image Velocimetry (PIV) is a technique which allows one to record images of large parts of flow fields in a variety of applications in gaseous and liquid media and to extract the velocity information out of these images [8]. The typical setting of a PIV experiment consists in the following components: the flow medium seeded with particles, droplets or bubbles, a double pulsed laser which illuminates the particles twice with a short time difference, a light sheet optics guiding a thin light plane within the flow medium, one or several CCD cameras which capture the two frames exposed by the laser pulses and a timing controller synchronising the laser and the camera. Once the flow motion has been captured, software tools are needed to evaluate and display the flow motion. The standard techniques work in a planar domain (2D-PIV), permitting estimation of the 2 planar components of the fluid motion (2C-PIV). The third spatial component can also be extracted using stereo techniques, dual-plane PIV or holographic recording (3C-PIV) [5]. The extension of the observation to a volume (3D-PIV) is currently an active area of research. To this end, multicamera configuration or holographic techniques (see [9]) has been proposed.

The most widely used technique for motion estimation in 2D-PIV is based on local correlation between two rectangular regions of the two images (see for instance [10]). This technique has a straightforward extension to 3D images. Another approach to motion estimation widely used in optical flow is a variational approach based on an energy minimisation where on the one hand, we assume the conservation of the intensity of the displaced objects (in our case the particles) and on the other hand, we assume a certain regularity of the obtained flow. A variational approach was proposed in [3] in the context of 2D PIV.

When we deal with 3D fluid flow estimation, physical constraints appear following the nature of the flow. Currently, an interesting challenging problem is to include such physical constraints (think for instance in the Navier-Stokes equations or incompressibility) into the 3D motion estimation model.

In this document we present an overview of the techniques we have developed for 3D motion estimation in the context of FLUID project. In particular we present extension of the correlation and standard variational techniques to 3D. We also present new 3D energy based method which preserves the incompressibility of the flow (i.e. \( \text{div}(\mathbf{u}) = 0 \)), that is, we include the incompressibility constraint in the energy-based method. In order to minimise the constrained energy we use a generalised Lagrange multiplier approach.

This document is organised as follows: in section 2, we describe the basic extensions of 2D fluid flow estimation methods to 3D. in section 3, we present the classic Helmholtz vector field decomposition that we use to analyse the constrained energy problem. In section 4, we describe an energy-based model constrained by the incompressibility of the flow. In section 5 we analyse the challenging nonlinear discrete optimisation problem that derives from the constrained energy. In section 6, we present some experiments on three 3D synthetic data sets, in section 7 we present some experiments on the real 3D PIV data provided by LaVision, and in section 8 we present an overview of the software we have implemented in the context of 3D PIV flow estimation.

2 Extension of 2D basic motion estimation techniques to 3D

2.1 Cross-Correlation

Cross-correlation is the most common technique for fluid motion estimation in PIV and is described, for instance, in [8]. We will denote \( I_1 \) and \( I_2 \) the two images from which we compute
the motion \( \mathbf{u} \), \( N \) the image dimension (in our case \( N = 3 \)) and \( \Omega \) the domain of definition of the images.

### 2.1.1 Basic Principle

Having the two volumes \( I_1 \) and \( I_2 \), for each voxel \( \mathbf{v} = (v_x, v_y, v_z) \) of \( I_1 \), the method takes a rectangular subvolume \( I_{1,v} \) of \( I_1 \) centered on \( \mathbf{v} \), and looks for a similar subvolume of \( I_2 \) centered on a neighbor \( \mathbf{v} + \mathbf{d} \) of \( \mathbf{v} \). The similarity measure between two rectangular subvolumes of the same dimensions is based on 2D cross-correlation and is defined as:

\[
C_\mathbf{v}(I_1, I_2)(\mathbf{d}) = \sum_{y = (a, b, c)} I_1(\mathbf{v} + y) I_2(\mathbf{v} + \mathbf{d} + y) \tag{1}
\]

The voxel \( \mathbf{v} \) is assigned the displacement \( \mathbf{d} \) which gives the maximal value of the cross-correlation. Doing this for every voxel in \( I_1 \) we obtain a complete vector field \( \mathbf{u} \).

![Figure 1: Correlation subvolumes](image)

### 2.1.2 Implementation using Fast Fourier Transform

Because the process of computing the cross-correlation for many subvolumes of \( I_2 \) and for each voxel is computationally heavy, the implementation takes advantage of the properties of the Fourier transform to improve the processing time. The Fourier transform has the property that a correlation in the spatial domain is equivalent to a multiplication in the Fourier domain.

\[
C_\mathbf{v}(I_1, I_2) = \mathcal{F}^{-1}(\hat{I}_{1,v} \hat{I}_{2,v}^*) \tag{2}
\]

where \( I_{1,v} \) is a rectangular subvolume of \( I_1 \) centered on the voxel \( \mathbf{v} \), \( \hat{I}_{1,v} \) is the Fourier Transform of the subvolume \( I_{1,v} \), the operator \(^*\) denotes the complex conjugate, and \( \mathcal{F}^{-1} \) denotes the inverse Fourier transform. The image \( C_\mathbf{v}(I_1, I_2)(\mathbf{d}) \) gives the result of cross-correlation for all displacements \( \mathbf{d} \) and the maximal value is a best estimate of the local displacement. Because of the hypothesis of periodicity introduced by the Fourier Transform, the window is usually chosen four times bigger than the expected displacement. The method is then extended to allow subvoxel accuracy by means of local interpolation of a Gaussian function close to the
When the correlation has been computed for every voxel, some kind of data validation procedure is needed to remove outliers.

Actually, we do not have to compute the correlation for each voxel, we can calculate the flow only for the voxels located on a given lattice. At the end of the process, we extrapolate the results and obtain a dense vector field. This improves not only the speed of the computation, but also in some cases the quality of the results because of the regularization induced by the extrapolation.

The whole process should be applied iteratively a few times using the current result as an initialization for the next iteration. The iterative process can be initialized with a null vector field \( u^0 = 0 \), and \( u_n+1 \) can be estimated at each voxel of the lattice using the displacement with maximal correlation for a window of \( I_2 \) displaced by \( u^n \):

\[
C_v(I_1, I_2, u^n) = F^{-1}(I_{1_v} \hat{I}_{2_v+u^n(v)})^*, \tag{3}
\]

By doing this, we can improve the accuracy of the fluid motion estimation. It also permits the progressive reduction of the size of the correlation window.

### 2.2 Variational Approach

Variational approaches to motion estimation are often used for optical flow computation \([6, 2, 1]\). It consists in minimizing an energy as a function of the displacement and that depends on a pair of images \( I_1 \) and \( I_2 \).

In this section, \( E \) will denote the energy functional to minimize. For a given 3D vector field \( u = (u^x, u^y, u^z)^t \), the norm of its gradient \( \| \nabla u \| \) is defined as \( \sqrt{\| \nabla u^x \|^2 + \| \nabla u^y \|^2 + \| \nabla u^z \|^2} \), and the Laplacian \( \Delta u = \text{div}(\nabla u) \) is defined as \( (\Delta u^x, \Delta u^y, \Delta u^z)^t \).

The energy to minimize is expressed as :

\[
E(u) = \int_\Omega \left( \underbrace{(I_1(x) - I_2(x + u(x)))^2}_\text{data term} + \underbrace{\alpha \| \nabla u(x) \|^2}_\text{regularization term} \right) dx, \tag{4}
\]

where \( \alpha \) is a scalar coefficient that weights the smoothing term. Under the assumption of intensity conservation for each voxel, the first term (data term) becomes zero when the first image matches the second one displaced by \( u \): \( I_1(x) = I_2(x + u(x)) \). This term tries to find the vector field that best fits the solution. The second term is a regularization term which smooths the vector field. There are a lot of ways to define the regularization term, including, for instance, discontinuities preserving constraints, etc. Since we deal with rather smooth flows we have used the simplest regularity term presented above.

Euler-Lagrange equations yield:

\[
(I_1(x) - I_2(x + u)).\nabla I_2(x + u) + \alpha \text{div}(\nabla u) = 0 \tag{5}
\]

#### 2.2.1 Numerical Scheme

We propose to look for the minimum of the energy by solving (5) directly using a fixed point approach. An alternative is to use a gradient descent with either explicit or semi-implicit scheme. We use an iterative method to find the vector field \( u \):

\[
\begin{cases}
  u^0 = u_0 \\
  u^{n+1} = u^n + h^{n+1}
\end{cases} \tag{6}
\]
where we update the vector field $\mathbf{u}$ at each iteration by adding another vector field $\mathbf{h}$ with small displacements. The displacement $\mathbf{h}$ being small, we can use first order Taylor expansions of $I_2$ and $\nabla I_2$ at $\mathbf{x} + \mathbf{u}^n$ to linearize (5), and we obtain:

$$dg - [gg^t - dH'] \mathbf{h} + \text{adiv}(\nabla \mathbf{u}^n + \nabla \mathbf{h}) = 0$$

(7)

denoting:

$$g(x) = \nabla I_2(x + \mathbf{u}^n)$$

(8)

$$d(x) = I_1(x) - I_2(x + \mathbf{u}^n)$$

(9)

$$H'(x) = H(I_2(x + \mathbf{u}^n)).$$

(10)

In the last equality, $H(I_2(x))$ denotes the Hessian matrix of $I_2$ at the location $\mathbf{x}$. The term in second order spatial derivatives is usually neglected, supposing that the image varies slowly. Then, (26) becomes:

$$dg + \text{adiv}(\nabla \mathbf{u}^n) - gg^t \mathbf{h} + \text{adiv}(\nabla \mathbf{h}) = 0$$

(11)

After discretization using finite differences, the operator $\text{div}(\nabla \mathbf{h})$ can be divided in two terms $-2N I \mathbf{h}$ and $S(\mathbf{h})$, where the $N$ is the image dimension and $I$ is the identity matrix. The first term only depends on values of $\mathbf{h}$ at the current position $\mathbf{x}$ and the second term only depends on values of $\mathbf{h}$ at neighbor positions of $\mathbf{x}$: the vector $S(\mathbf{h})$ is written:

$$S(\mathbf{h}) = \left( \sum_{y \in N^*(x)} h_x^y(y), \sum_{y \in N^*(x)} h_y^y(y), \sum_{y \in N^*(x)} h_z^y(y) \right),$$

(12)

where $N^*(x)$ denotes the direct neighbors of $\mathbf{x}$ (4 in 2D and 6 in 3D), and $\mathbf{h} = (h_x, h_y, h_z)^t$.

Using $\mathbf{h}^{n+1}$ for the current location $\mathbf{x}$ and $\mathbf{h}^n$ for its neighbors, (11) becomes:

$$A\mathbf{h}^{n+1} = b$$

(13)

with $A = gg^t + \alpha 2N I$, and $b = dg + \text{adiv}(\nabla \mathbf{u}^n) + S(\mathbf{h}^n)$. The matrix $A$ is real, symmetric and positive definite, so it can be inverted and we can compute for each position $\mathbf{x}$, $\mathbf{h}^{n+1} = A^{-1}b$. To improve the convergence rate, we use a Gauss-Seidel method which updates the displacement $\mathbf{h}^{n+1}$ at position $\mathbf{x}$ using the values of $\mathbf{h}^{n+1}$ already calculated. This scheme is recursive and to avoid privileging the direction of scanning the image, we apply two successive iterations of Gauss-Seidel in reverse directions. Furthermore, we use a pyramidal approach to compute the displacement flow at several scales, using the results from a given scale to initialize to the following higher scale.

### 3 Helmholtz vector field decomposition

The results presented in this section can be found in any classical book on fluid mechanics (see for instance [4, 11, 12]). First, we introduce the following notations:

$$H_s \equiv \{ \mathbf{u}_s \in C^1(\Omega) \cap C(\overline{\Omega}) : \text{div}(\mathbf{u}_s) = 0 \text{ in } \Omega \text{ and } \mathbf{u}_s \cdot \mathbf{n} = 0 \text{ in } \partial\Omega \}$$

(14)

where $\mathbf{n}(\mathbf{x})$ is the vector normal to the boundary of $\Omega$.

$$H_r \equiv \{ \mathbf{u}_r \in C^1(\Omega) \cap C(\overline{\Omega}) : \exists \ p \in H^1(\Omega) \text{ such that } \mathbf{u}_r = \nabla p \text{ in } \Omega \}$$

(15)

where $H^1(\Omega)$ is the usual Sobolev space.
Theorem 1 (Helmholtz decomposition). Let $u \in C^1(\Omega) \cap C(\bar{\Omega})$ a 3D vector field, then there exist $u_s \in H_s$, $u_r \in H_r$

$u = u_s + u_r$

$u_s$ and $u_r$ are orthogonal in $L^2(\Omega)$, i.e.

$$\int_{\Omega} u_s(x) u_r(x) dx = 0$$

and $u_r = \nabla p$ where $p$ is a solution of the Poisson type equation:

$$\begin{align*}
\Delta p &= \text{div}(u) & \text{in } \Omega \\
\frac{\partial p}{\partial n} &= u \cdot n & \text{in } \partial \Omega.
\end{align*}$$

We observe that the above Poisson’s equation allows us to estimate the projection of $u$ in $H_r$ and $H_s$. Indeed, given $p(.)$ the solution of the above Poisson equation, we can define the following projection operators in the space $H_r$ and $H_s$:

$$P_r(u) \equiv u_r = \nabla p$$

$$P_s(u) \equiv u_s = u - \nabla p$$

4 Incompressibility constrained energy model

A large class of fluid flows satisfies the incompressibility constraint (i.e. $\text{div}(u) = 0$). In this section, we focus on how to include such incompressibility constraint in the 3D flow motion estimation. To this end, we look for local minima of the constrained energy

$$\min_{u \in H_s} E(u),$$

where $E(u)$ is defined in (4) and $H_s$ is defined in (14). First we will show a generalisation of Lagrange multipliers technique applied to the constrained energy problem (19).

Theorem 2 (Lagrange multipliers generalisation). Let $u \in H_s$ a local minimum of the constrained energy (19), then

$$\nabla E(u) \in H_r,$$

where $H_r$ is defined in (15).

**Proof:** We will show that if $\nabla E(u) \notin H_r$ then $u$ can not be a local minimum of the constrained energy (19). Indeed, if $\nabla E(u) \notin H_r$ then by theorem 1, we can decompose

$$\nabla E(u) = u_s + u_r,$$

where $u_s \in H_s$, $u_r \in H_r$ and $u_s \neq 0$. Next, we consider the function

$$\phi(t) = E(u + tu_s),$$

since $u_s, u \in H_s$, then $u + tu_s \in H_s$. An straightforward computation yields to

$$\phi'(0) = \int_{\Omega} \nabla E(u)(x) \cdot u_s(x) dx.$$
Therefore, using (21) and theorem 1 we obtain
\[ \phi'(0) = \int_{\Omega} (u_s(x) + u_r(x)) \cdot u_s(x) \, dx = \int_{\Omega} u_s^2(x) \, dx > 0. \]
and we conclude that \( u \) can not be a minimum of the constrained problem (19).

Remark 1. The above theorem can be considered as a generalisation of the usual Lagrange multipliers result where the gradient of the constrained energy has to be in the direction orthogonal to the constraint surface. Theorem 2 clarifies the theoretical structure of local minima of the constrained energy (19) and it will be useful to design algorithms to estimate local minima of the constrained energy as shown in the next section.

Remark 2. We observe that condition (20) is equivalent to the equation
\[ \nabla E(u) = P_r(\nabla E(u)). \quad (22) \]
In the case of the standard Lagrange multipliers technique, the above equation is equivalent to the usual Lagrange multipliers condition
\[ \nabla E(u) = \lambda \nabla g(u), \]
where \( g(u) = 0 \) would be the constraint. In this case \( P_r \) would represent the projection of \( \nabla E(u) \) in the orthogonal space to the tangent space to the level set \( g(u) = 0 \).

Remark 3. Theorem 2 is general and can be applied to any energy \( E(u) \) constrained by the incompressibility condition, so it does not depend on the particular choice of the energy given in (4).

Remark 4. In case we are looking for irrotational flow \( (\text{rot} \, u = 0) \) and we constrain the energy to irrotational flow, we can easily adapt Theorem 2. We can show using the same technique that if \( u \) is a local minimum of the energy in the space of irrotational flow then \( \nabla E(u) \) belongs to the space of divergence free flow.

5 Nonlinear discrete optimisation problem.

In order to solve numerically the constrained energy (19), we discretize such constrained energy in the following way: \( \Omega \) will be a discrete rectangular volume, \( \hat{u} \equiv \{\hat{u}_{i,j,k}\} \) will be an approximation of \( u(i \cdot \delta x, j \cdot \delta y, k \cdot \delta z) \), where \( u \) is a local minimum of (19) and \( (\delta x, \delta y, \delta z) \) is the voxel size.

Next, we discretize equation (22) and we arrive to the discrete nonlinear equation to solve :
\[ \nabla E(\hat{u}) = P_r(\nabla E(\hat{u})). \quad (23) \]
This is a challenging nonlinear system of equations. In order to solve it, we will use an iterative scheme. Let \( \hat{u}^n \) be an estimation of a local minimum at step \( n \). We write \( \hat{u}^{n+1} \) as
\[ \hat{u}^{n+1} = P_s(\hat{u}^n + h), \quad (24) \]
where \( P_s \) is given by (18) and \( h \) is the unknown in each iteration. Following equation (23), in order to compute \( h \), we solve the equation
\[ \nabla E(\hat{u}^n + h) = P_r(\nabla E(\hat{u}^n)) \]
To simplify the above equation we linearise $\nabla E(\hat{u}^n + h)$ from (5) and we solve:

$$
\begin{align*}
&((I_1(x) - I_2(x + \hat{u}^n) - h^T \nabla I_2(x + \hat{u}^n))\nabla I_2(x + \hat{u}^n) \\
&+ \alpha \text{div}(\nabla \hat{u}^n) + \text{div}(\nabla h)(x) = P_r(\nabla E(\hat{u}^n)))
\end{align*}
$$

(25)

we use standard finite difference schemes to discretize the differential operators presented in the above equation. We observe that the above equation is linear in $h$ and therefore it can be solved using standard techniques. We can write the above equation in a compact way as

$$- [gg^T] h + \text{div}(\nabla h) = P_r(\nabla E(\hat{u}^n)) - dg - \alpha \text{div}(\nabla \hat{u}^n),$$

(26)

where

$$g(x) = \nabla I_2(x + \hat{u}^n)$$

(27)

$$d(x) = I_1(x) - I_2(x + \hat{u}^n).$$

(28)

Once the system of equations (26) is discretized, it becomes linear.

After discretization using finite differences, the operator $\text{div}(\nabla h)$ can be divided in two terms $-2N I_1 h$ and $S(h)$, where the $N$ is the image dimension and $I$ is the identity matrix. The first term only depends on values of $h$ at the current position $x$ and the second term only depends on values of $h$ at neigbour positions of $x$: the vector $S(h)$ is written:

$$
S(h) = \begin{pmatrix}
\sum_{y \in N^*(x)} h^x(y) \\
\sum_{y \in N^*(x)} h^y(y) \\
\sum_{y \in N^*(x)} h^z(y)
\end{pmatrix},
$$

(29)

where $N^*(x)$ denotes the direct neigbours of $x$ (6 in 3D), and $h = (h^x, h^y, h^z)^T$.

We write the system of equations (26) as a fixed point equation in the following way:

$$A h = b + S(h),$$

(30)

where

$$A = gg^T + \alpha 2N \cdot I,$$

$$b = dg + \alpha \text{div}(\nabla \hat{u}^n) - P_r(\nabla E(\hat{u}^n)) + S(h^n)$$

In order to solve the above equation, we use Gauss-Seidel numerical scheme. Furthermore, in order to increase the convergence speed and to avoid spurious local minima, we use an standard pyramidal approach to compute the displacement flow at several scales, using the results from a given scale to initialise to the following higher scale. Finally, once $h$ is computed, we estimate $\hat{u}^{n+1}$ using the projection $P_s$ (24).

To resume, the algorithm consists in estimating $\hat{u}^{n+1}$ from $\hat{u}^n$ iteratively until convergence, applying the following steps:

1. Computation of $P_r(\nabla E(\hat{u}^n))$,
2. Computation of $h$ by solving the linear system (26),
3. Computation of $\hat{u}^{n+1} = P_s(\hat{u}^n + h)$.

The projection operators $P_s(.)$ and $P_r(.)$ defined in (18) and (17) are computed from the linear equation (16) using standard numerical techniques.
6 Experiments and Results on Synthetic images

In this section, we present experiments on synthetic data using both correlation and variational methods. We use 3D flows based on realistic flow models to check the performance of the proposed methods. In these experiments, we first apply the standard correlation or variational methods to obtain a good approximation of the flow and then we refine the results with the new variational approach.

6.1 Choice of the parameters

The cross-correlation parameters are the window size in each dimension and the lattice spacing. The window size is approximately set to four times the expected maximal displacement and is the same in each dimension. In the following experiments, we use a lattice spacing of 2 voxels in each dimension, and the final result is interpolated to obtain a dense estimation. The variational approach uses the parameters $\alpha$ and the number of scales for the pyramidal approach. In the following experiments, we set $\alpha$ to 0.5 for both the standard and the constrained variational approaches.

6.2 Description of the models

In the first model (Fig. 2, top left), we use an incompressible 3D flow model suggested by Professor F. Scarano that can be found in [13] (section 3-9.2). It corresponds to the Stokes’s solution for an immersed sphere. The flow moves in the horizontal axis direction with a velocity $(U, 0, 0)$, and it avoids the sphere located at the center of the volume. The flow inside the sphere is null.

Having a sphere with radius $\alpha$ and center $(0, 0, 0)$, and a 3D point $(x, y, z)$ at a distance $r$ from the sphere center, the flow outside the sphere follows:

$$u = U \left(1 - \frac{3\alpha}{4r^3}(2x^2 + y^2 + z^2) + \frac{\alpha^3}{4r^5}(2x^2 - y^2 - z^2)\right)$$

$$v = U \left(-\frac{3\alpha}{4r^3}xy + \frac{3\alpha^3}{4r^5}xy\right)$$

$$w = U \left(-\frac{3\alpha}{4r^3}xz + \frac{3\alpha^3}{4r^5}xz\right).$$

The average displacement in the first model is 1.2847 voxels.

The two other models (Fig. 2, top right and bottom) have been provided by the Cemagref Rennes, France and they have been obtained using a Large Eddy Simulation of the incompressible Navier-Stokes equations which defines the turbulent motion after a cylinder [7]. They simulate a volume with synthetic particles following the horizontal axis and a cylinder located on the $z$-axis obstructing the flow perpendicularly. We use two successive images from these sequences.

The second model is a volume of $960 \times 960 \times 144$ voxels, but we limit our experiments to a window of $256 \times 64 \times 64$ voxels to reduce the computation time. This window includes part of the cylinder and the turbulence behind it. The average displacement in this model is 0.8865 voxels.

The third model is similar to the second one with a volume of size $256 \times 128 \times 144$ voxels. It has a lower density of particles which is more realistic given the state of the art in 3D-PIV techniques. Moreover, the average displacement of the flow is large: 5.0396 voxels. As
Figure 2: Top left, model 1: sphere. Top right, model 2: cylinder. Bottom, model 3: cylinder with lower particle density and larger flow motion.
a consequence, this model is more realistic and more challenging in the context of 3D motion estimation.

In order to evaluate the quality of our tests, we compute the following statistics: the Euclidean error (AEE, eq. 32) and the angular error (AAE, eq. 33) for these sequences according to [1].

\[
AEE = \frac{1}{N} \sum_{i=1}^{N} \sqrt{\|u_i - u_{ref_i}\|^2},
\]

\[
AAE = \frac{180}{N\pi} \sum_{i=1}^{N} \arccos\left(\frac{u_i u_{ref_i} + 1}{\sqrt{\|u_i\|^2 + 1}\sqrt{\|u_{ref_i}\|^2 + 1}}\right),
\]

where \(u_i\) is the estimated vector and \(u_{ref_i}\) is the known ground truth in voxel \(i\).

6.3 Experiments with model 1 (sphere)

Table 1 shows the average Euclidean error (AEE) and average angular error (AAE), with their standard deviations (STD), reached for the cross-correlation and the variational methods before and after the refinement. The error at each voxel is computed as the magnitude of the difference between the ideal displacement and the estimated one, and is measured in voxel unit. The correlation was applied 11 times with a window size of 8 voxels. The individual variational approach was applied using \(\alpha = 0.5\) and 3 scales. The mean error is approximately divided by two after applying the refined variational approach, and the initialisation with correlation gives a better result than the initialisation with a variational approach. Figure 3 (right) shows the final average error distribution using the cross-correlation followed by the refined variational approach. We can observe that the highest error is located at the sphere boundaries.

![Figure 3: Left, real flow (with zoom). Right, final error distribution using cross-correlation followed by constrained energy minimisation.](image)

The left curve in Fig. 4 displays the average error evolution using the combined scheme (using cross-correlation followed by constrained energy minimisation). First, we apply 11 iterations of correlation technique (we observe that the correlation reaches a stable average error after 11 iterations). Next, we use the output flow provided by the correlation as the input flow of the
Table 1: Comparison of the different methods for model 1. Average displacement = 1.2847.

<table>
<thead>
<tr>
<th>Method</th>
<th>AEE (± STD)</th>
<th>AAE (± STD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corr.</td>
<td>0.0290 ± 0.0296</td>
<td>0.9214 ± 1.3560</td>
</tr>
<tr>
<td>Corr.+Incomp.</td>
<td>0.0139 ± 0.0158</td>
<td>0.4413 ± 0.7451</td>
</tr>
<tr>
<td>Var.</td>
<td>0.0293 ± 0.0319</td>
<td>0.9124 ± 1.4700</td>
</tr>
<tr>
<td>Var.+Incomp.</td>
<td>0.0196 ± 0.0284</td>
<td>0.6193 ± 1.2073</td>
</tr>
</tbody>
</table>

refined variational technique (curve after iteration 11). We observe a significant improvement in the flow estimation error after using the proposed refined variational method.

Figure 4: Left, average Euclidean error evolution using the combined scheme (11 times cross-correlation + constrained variational).

6.4 Experiments with model 2 (cylinder)

We ran the same experiments for this model. Table 2 shows the average Euclidean error (AEE) and average angular error (AAE), with their standard deviations (STD), reached for the cross-correlation and the variational methods before and after the refinement. The correlation was applied 6 times with a sequence of different window sizes: 16, 16, 8, 8, 4, 4. The variational approach was applied using $\alpha = 0.5$ and 3 scales. Finally, the constrained variational method was applied with $\alpha = 0.5$ and one scale. In this experiment, the variational and the correlation methods alone reach similar accuracies, and after the refinement, the cross-correlation reaches a slightly better result. In both cases, the new refined variational approach reduces the mean error by at least half. Figure 5 (bottom) shows the final average error distribution using the combination of the cross-correlation and the refined variational schemes. As in the previous model, the highest error is also located at the obstacle boundaries.

The curve in Fig. 6 displays the average error evolution using the combined scheme of correlation and refined variational approaches. It shows that the correlation reaches a stable
average error after 6 iterations and that an additional iteration of the proposed variational approach reduces considerably the mean error.

6.5 Experiments with model 3 (cylinder with low particle density)

This third model defines a large turbulence motion with a low particle density. The parameters used in this test are the same than the previous ones. The correlation was applied 6 times with a sequence of different window sizes: 16, 16, 8, 8, 4, 4. The variational approach was applied using \( \alpha = 0.5 \) and 3 scales. The refined variational method was applied with \( \alpha = 0.5 \) and one scale. In table 3, it is shown the AEE and AAE, with their STD’s, reached for the cross-correlation and the variational methods before and after the refinement.

In this test, the variational and the correlation methods alone reach similar accuracies. After the refinement, as it happened in the previous tests, the cross-correlation reaches a slightly better result. In both cases, the new refined variational approach reduces the mean error over 10% percent. In Fig. 7 top, the real flow is shown. Fig. 7 bottom shows the final average error distribution using the combination of the cross-correlation and the constrained variational schemes. The highest errors are also located at high turbulence regions where there are not many particles. Thus, it is more difficult to obtain accurate results in those regions.
Figure 6: Average error evolution using the combined scheme (6 times cross-correlation + constrained variational).

Table 3: Comparison of the different methods for model 3. Average displacement = 5.0396.

<table>
<thead>
<tr>
<th>Method</th>
<th>AEE (± STD)</th>
<th>AAE (± STD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corr.</td>
<td>0.6265 ± 0.8797</td>
<td>7.6469 ± 12.6845</td>
</tr>
<tr>
<td>Corr.+Incomp.</td>
<td>0.5480 ± 0.6893</td>
<td>6.8157 ± 10.5740</td>
</tr>
<tr>
<td>Var.</td>
<td>0.7003 ± 0.9162</td>
<td>8.7224 ± 14.6265</td>
</tr>
<tr>
<td>Var.+Incomp.</td>
<td>0.6046 ± 0.8263</td>
<td>7.5359 ± 12.4671</td>
</tr>
</tbody>
</table>
Figure 7: Top, real flow. Bottom, final error distribution using cross-correlation followed by constrained energy minimisation.
7 Experiments and Results on LaVision Real Image dataset

In this section we present some numerical experiments on LaVision real image dataset. Since the real flow is unknown, what we have done is to compare the 3D motion estimation obtained by LaVision FLUID partner and the results obtained by the AMI Group with the combination of the correlation method and PDE incompressibility constrained energy based method which seems to be the most accurate technique (at least in the synthetic experiments). As we show in the numerical experiments the obtained results using both approaches are coherent. As we illustrate in the images we present in this section, by visual inspection we observe that the shape of the flow are quite similar and on the other hand the statistics we have estimated between both 3D flows show that in fact the flows are close. This is a very promising result because it is a good sign about the accuracy of the estimation methods.

**Flow estimation statistics:**

<table>
<thead>
<tr>
<th></th>
<th>LaVision</th>
<th>AMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average displacement magnitude</td>
<td>9.3629</td>
<td>9.3702</td>
</tr>
<tr>
<td>Average of Euclidean error</td>
<td>0.1902</td>
<td>0.1660</td>
</tr>
<tr>
<td>Average of angular error</td>
<td>1.1106</td>
<td>1.4062</td>
</tr>
</tbody>
</table>

![Figure 8: 3D PIV LaVision Data Set](image-url)
Figure 9: 3D flow estimation obtained by LaVision (Correlation technique)

Figure 10: 3D flow estimation obtained by AMI Group (Correlation + PDE incompressibility constraint)
Figure 11: Comparison between LaVision 3D flow estimation (in green) and the one obtained AMI Group (in red)

Figure 12: Illustration of the Euclidean error between LaVision 3D flow estimation and the one obtained AMI Group
Figure 13: Illustration of the angular error between LaVision 3D flow estimation and the one obtained AMI Group
8 Software for 3D motion estimation

8.1 C/C++ Interface

We have developed a library in C/C++ called amifluid which allows us to compute the motion between two 3D volumes. The main functions of this library are designed to obtain the estimation using even the cross-correlation technique or the variational approach.

The ami_optic_flow_correlation_3d function computes the motion estimation using the cross-correlation technique. It has to be called iteratively using the previous result as the initialization of the next iteration. It uses the Fast Fourier Transform to save computation time, but this implementation needs the window size to be power of two in each dimension. The window size should be four times the maximal displacement expected so the usual way to call this function is with a big window size first and then reducing it for more accuracy in the following iterations.

The ami_horn_schunck_optic_flow_3d function solves the same problem but using the basis variational approach instead. It uses a pyramidal approach and iterates by itself so it only needs to be called one time with an initial flow estimation which can be zero for all points or the result of another technique.

8.2 AMILab Interface

In the practice we use the AMILab software (http://serdis.dis.ulpgc.es/~krissian/HomePage/Software/AMILab/), which allows the easy visualization of images, 3D volumes, vector fields, etc. This software has also its own scripting language and permits the wrapping of C/C++ functions so we can call them from its environment and visualize the results in a very simple way. We have also developed the wrapping of the main library functions to use them in the context of AMILab scripting.

In order to use the AMILab wrapping it is necessary to import the ami library putting import ami at the beginning of the scripts. Using the AMILab scripting language, it is also possible to design graphical interfaces in order to be able to set the parameters and to call the different functions easier. An example of these scripts can be seen in the Appendix and the interfaces developed can be seen in Figure 14.

![Figure 14: Left, correlation interface. Right, PDE interface](image)

The amiOFCorr3D function wraps the motion estimation using the cross-correlation technique
and the amiOFPDE function allows to do the same but using the variational approach.

### 8.3 ami_optic_flow_correlation_3d

This C function computes the displacement between two 3D images (image_1 and image_2) using the Cross-Correlation method on a given lattice.

```c
void ami_optic_flow_correlation_3d(float ***image_1, float ***image_2,
                                  int dim_x, int dim_y, int dim_z,
                                  ami_v3f ***u,
                                  ami_v3f p_init,
                                  ami_v3f p_disp,
                                  int size_correlation_window[3],
                                  float window_deformation)
```

- `image_1, image_2`: image pair (input data)
- `dim_x, dim_y, dim_z`: image dimensions (input data)
- `u`: input/output flow (3D matrix of 3D vectors), it has to be initialized
- `p_init`: initial point of the lattice
- `p_disp`: displacement to compute the lattice
- `size_correlation_window`: size of the correlation window (it must be power of 2)
- `window_deformation`: window parameter to control the warping in the correlation window.
  - 0: image_2 is warped using flow in all points of the window
  - 1: image_2 is warped using flow in the central point of the window

### 8.4 ami_optic_flow_correlation_3d_new

This C function computes the displacement between two 3D images (image_1 and image_2) using the Cross-Correlation method on a given lattice. The differences with the previous correlation function are:

- images can be even integer or float (template)
- vector field adjusted to the lattice points requiring less memory
- faster computation using threads

```c
void ami_optic_flow_correlation_3d_new(T ***image_1, T ***image_2,
                                     int dim_x, int dim_y, int dim_z,
                                     ami_v3f ***u,
                                     ami_v3f p_init,
                                     ami_v3f p_disp,
                                     int size_correlation_window[3],
                                     int num_threads)
```

- `image_1, image_2`: image pair (input data)
- `dim_x, dim_y, dim_z`: image dimensions (input data)
- `u`: input/output flow (3D matrix of 3D vectors), it has to be initialized and to be adjusted to the lattice size
- `p_init`: initial point of the lattice
- `p_disp`: displacement to compute the lattice
- `size_correlation_window`: size of the correlation window (it must be power of 2)
- `num_threads`: number of threads to throw in parallel
8.5 ami_horn_schunck_optic_flow_3d

This C function computes the displacement between two 3D images (image_1 and image_2) using the basic variational approach. The vector field computed is dense having the same size as the images.

```c
int ami_horn_schunck_optic_flow_3d(float ***image_1, float ***image_2,
    ami_v3f ***u,
    const PDEOpticFlowParam& params,
    int dim_x, int dim_y, int dim_z)
```

- **image_1, image_2**: image pair (input data)
- **u**: input/output flow (3D matrix of 3D vectors), it has to be initialized
- **params**: PDE specific parameters detailed in \ref{tab:PDEParams}
- **dim_x, dim_y, dim_z**: image dimensions, they have to be a multiple of zoomfactor[i]^Nscales

8.6 PDEOpticFlowParam

Parameters structure for the ami_horn_schunck_optic_flow_3d(.) function

- **Phi**: Smoothing function. Possible values are:
  - phi_x: phi(x)=x, standard minimization of the square of the gradient norm
  - phi_sqrt_x: phi(x)=sqrt(x+epsilon), square root normalization
  Default value is phi_x.

- **GradComp**: Gradient computation. Possible values are:
  - grad_max: gradient is calculated using the maximal absolute value with the neighbors in each direction
  - grad_inv: gradient is calculated with a \texttt{local} 3x3 mask (only 2D), which is invariant under rotation as much as possible
  Default value is \texttt{grad\_max}.

- **global_constraint**: If the smoothing constraint is applied to the total displacement or to the additional displacement. Default value is 1.
- **smooth_gradient**: If the gradient is computed on the smoothed image. Default value is 1.
- **symmetric**: Symmetric formulation. Default value is 0.
- **beta[3]**: Parameter of the initial optical flow function smoothing coefficients (like Gaussian convolution). Default value is 0.25 for the three dimensions.
- **ami_v3f alfa**: Coefficient for the smoothing term of the energy. Default value is 2.0 for the three dimensions.
- **zoom_factor[3]**: Zoom factor in each dimension. Default value is 2 for the three dimensions.
- **Nscales**: Number of scales in the Pyramidal computation. Default value is 3.
- **TOL_Scales**: Tolerance threshold for the scale iterations. Default value is 1E−2.
- **gradient_type**: Different way to calculate the gradient of the energy. Default value is 0.
8.7 amiOFCorr3D

This is the wrapped function to call ami_optic_flow_correlation_3d (8.3) from AMILab.

```c
amiOFCorr3D(im1, im2, &u, wsx, wsy, wsz, windef,
    stepx, stepy, stepz,
    initx, inity, initz)
```

- `im1, im2`: image pair (input data)
- `u`: input/output flow, it has to be initialized
- `wsx, wsy, wsz`: size of the correlation window (power of 2), default value is (32,32,32)
- `windef`: window deformation, default is 1
- `stepx, stepy, stepz`: displacement grid, default is (4,4,4)
- `initx, inity, initz`: initial point, default is (0,0,0)

8.8 amiOFCorr3DNew

This is the wrapped function to call ami_optic_flow_correlation_3d_new (8.4) from AMILab.

```c
amiOFCorr3DNew(im1, im2, &u, wsx, wsy, wsz,
    stepx, stepy, stepz,
    initx, inity, initz,
    num_threads)
```

- `im1, im2`: image pair (input data)
- `u`: input/output flow, it has to be initialized and has the exact number of pixels of the lattice
- `wsx, wsy, wsz`: size of the correlation window (power of 2), default value is (32,32,32)
- `stepx, stepy, stepz`: displacement grid, default is (4,4,4)
- `initx, inity, initz`: initial point, default is (0,0,0)
- `num_threads`: number of threads to throw in parallel, default is 2

8.9 amiOFPDE

This is the wrapped function to call horn_schunck_optic_flow_3d (8.5) from AMILab.

```c
amiOFPDE(im1, im2, &u,
    betax, betay, betaz, zoomx, zoomy, zoomz,
    alfaz, alfay, alfaz, Conv_GS, Nscales,
    Conv_Scales, gradtype, smoothtype)
```

- `im1, im2`: image pair (input data)
- `u`: input/output flow, it has to be initialized
- `betax, betay, betaz`: initial optical flow function smoothing coefficients (like Gaussian convolution), default: (0.25,0.25,0.25)
- `zoomx, zoomy, zoomz`: zoom factor, default: (2,2,2)
- `alfaz, alfay, alfaz`: coefficients for the smoothing term of the energy, default: (4,4,4)
- `Conv_GS`: tolerance for Gauss Seidel scheme, default value is 1E−5
- `Nscales`: number of scale for pyramidal approach, default is 3
Conv\_Scales: tolerance for each scale, default is 1E−5

tolerance

Gradtype: gradient type, with value between 0 and 3, default 2

- 0: gradient of warped image
- 1: upwind method following the sign of image1 and warped image2
- 2: average of the gradient of image1 and warped image2
- 3: using the hessian matrix

Smoothtype: smoothing function type, default 0

- 0: standard function \( \phi(x) = x \)
- 1: \( \phi(x) = \sqrt{x+\epsilon} \)

8.10 3D Warping

```c
void ami_warpimage_3df(float ***image, ami_v3f ***u, float ***image_output, int dim_x, int dim_y, int dim_z, const PDEOpticFlowParam & param)
```

- `image`: input 3D image
- `u`: 3D dense motion vector field
- `image_output`: output 3D image
- `dim_x`, `dim_y`, `dim_z`: image dimensions
- `param`: FlowEstimationParameters

8.11 3D flow estimation with the incompressibility constraint

```c
int ami_incompressible_optical_flow_3d(float ***image_1, float ***image_2, ami_v3f ***u, const PDEOpticFlowParam & param, int dim_x, int dim_y, int dim_z);
```

- `image_1`, `image_2`: 3D input image pair to estimate the flow.
- `u`: output estimated flow.
- `param`: input optic flow parameters.
- `dim_x`, `dim_y`, `dim_z`: input image dimension.

8.12 Solenoidal projection

```c
int ami_solenoidal_proyeccion_3d(ami_v3f ***u, float beta[3], int zoom_factor[3], float TOL_GaussSeidel, int Nscales, double A, int dim_x, int dim_y, int dim_z)
```

- `zoom_factor[3]`: ZOOM FACTOR IN EACH DIMENSION IT HAS TO BE 1 OR 2
- `TOL_GaussSeidel`: CONVERGENCE PARAMETER OF GAUSS SEIDEL SCHEME
- `Nscales`: NUMBER OF RESOLUTIONS IN THE PYRAMIDAL APPROACH
- `A`: COEFFICIENT FOR CONVERGENCE OF LAPLACE EQUATION
- `dim_x`, `dim_y`, `dim_z`: IMAGE DIMENSIONS

8.13 Example of Graphical Interface in AMILab

```c
w2 = ParamWin(“PDE_Optical_Flow”)
```

- `beta_id` = w2.AddFloat(beta,0.01,5)
- `alfa_id` = w2.AddFloat(alfa,0.01,20)
Conv\_GS\_exp\_id = w2.AddInt(Conv\_GS\_exp, -10, 0)
Nscales\_id = w2.AddInt(Nscales, 1, 5)
Conv\_Scales\_exp\_id = w2.AddInt(Conv\_Scales\_exp, -5, 0)
gradtype\_id = w2.AddInt(gradtype, 0, 7)

restart\_id = w2.AddButton("Restart", CB\_Restart\_pde)
iterate\_id = w2.AddButton("Iterate", CB\_Iterate\_pde)
w2.CreateWin
w2.Display

References


