Published in Image Processing On Line on 2019–03–08.
Submitted on 2018–11–20, accepted on 2019–02–15.
ISSN 2105–1232 © 2019 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2019.238

An Analysis and Speedup of the FALDOI Method for Optical Flow Estimation

Ferran P. Gamonal[1], Coloma Ballester[1], Gloria Haro[1], Enric Meinhardt-Llopis[2], Roberto P. Palomares[1]

[1] DTIC, Universitat Pompeu Fabra, Barcelona, Spain
{ferran.perez, coloma.ballester, gloria.haro, roberto.palomares}@upf.edu
[2] CMLA, ENS Paris-Saclay, France
enric.meinhardt@cmla.ens-cachan.fr

Communicated by Luis Álvarez and Jose-Luis Lisani *Demo edited by* Jose-Luis Lisani

Abstract

We present a detailed analysis of FALDOI, a large displacement optical flow method proposed by P. Palomares et al. This method requires a set of discrete matches, which can be extremely sparse, and an energy functional which locally guides the interpolation from the matches. It follows a two-step minimization method at the finest scale which is very robust to the outliers of the sparse matcher and can capture large displacements of small objects. The results shown in the original paper consistently outperformed the coarse-to-fine approaches and achieved good qualitative and quantitative performance on the standard optical flow benchmarks. In this paper we revise the proposed method and the changes done to significantly reduce its execution time while reporting nearly the same accuracy. Finally, we also compare it against the current state-of-the-art to assess its performance.

Source Code

The C/C++ source code and its documentation are available at the IPOL web page of this article[1]. Program usage and compilation details are described in the `README.md` file. Python scripts to handle the calls to the binary files are also provided. If you need to report bugs, issues or have any doubts about the source code, please, open an issue on the GitHub repository https://github.com/fperezgamonal/faldoi-ipol.

Supplementary Material

We also attach a compressed file containing some auxiliar functions used to compute metrics, generate random subsets or partitions from the MPI-Sintel dataset and convert *flo* files to *png*. Additionally, we attach the full set of images included in the paper results and their output flow files computed with FALDOI. You will find a `README.txt` in each directory that explains how to use this material. If you run into any problems, please contact the e-mail address provided in the `README` files mentioned above.

Keywords: optical flow; variational methods; coordinate descent methods; sparse matches; parallelization

[1] https://doi.org/10.5201/ipol.2019.238

1 Introduction

In this work the problem of optimal flow estimation with variational approaches is addressed. Optical flow is the apparent motion field between two consecutive frames of a video. More generally, it can be defined as a dense correspondence field between an arbitrary pair of images.

We present a thorough description and a faster implementation of FALDOI [23], a strategy to compute good local minima of any optical flow energy functional which is able to capture large displacements. FALDOI stands for *Large Displacement Optical Flow method by an Astute Initialization.* The method relies on a discrete set of matches between two input images, which can be extremely sparse and contaminated by outliers. These matches are used as a guide to find a local minimum of the energy functional. The novelties include how these matches are used in the optimization problem. The method is a two-step minimization process of the optical flow energy. The first step computes a good and dense local minimum by propagating the initial matches with a region growing strategy. The propagation is driven by the minimization of the energy on patches and the order of update is established by the value of the local version of the energy, i.e. the energy restricted to a square patch of the image domain. In particular, given a priority queue where all candidates are stored, we select the one with the lowest energy, minimize over the patch centered at it, insert the neighboring pixels to the queue ranked according to the patch energy value and select again the candidate with the lowest energy. This process is repeated iteratively as we will see in more detail in Section 2. In this work, we analyze the method's features in depth and propose an optimization scheme to reduce the execution time while maintaining the same estimation accuracy. All the results included in this paper have been computed with the updated version of the code.

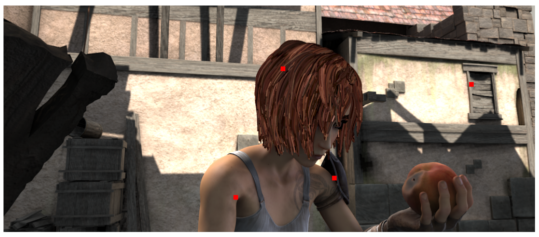
The remainder of the paper is organized as follows. Section 2 describes the FALDOI method. Section 3 explains in detail the changes made to the code to significantly reduce its execution time while maintaining the same estimation error. Section 4 presents results for several energy functionals and an analysis of the properties and performance of FALDOI depending on the density of the initial set of discrete matches and selected parameters. We also include comparisons between employing the speedup optimization or not in terms of error, run-time and qualitative evaluation. Finally, the conclusions are summarized in Section 5.

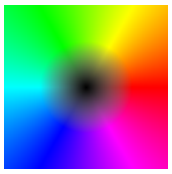
2 Proposed Minimization Strategy

In this section we revise the main features of the FALDOI minimization strategy. The strategy is compatible with any optical flow energy functional and is based on estimating a good local minimum from a discrete set of matches (computed via Algorithm 2). It benefits both from the sparse techniques, which handle arbitrarily large displacements, and from the continuous optimization of a variational formulation, which yields dense flow fields with subpixel accuracy. The method assumes that at least some matches are correct and propagates the correct information from those initial correspondences (referred to as *seeds*) by minimizing the energy around them. A challenging situation, where each region of smooth movement has *only* one correct seed, is illustrated in Figure 1. In particular, the optical flow has been obtained from four seeds; one on the head, arm, shoulder pad and background, respectively (shown in red in Figure 1), which have been chosen randomly (inside a limited area for each targeted image object) from the ones computed by SIFT.

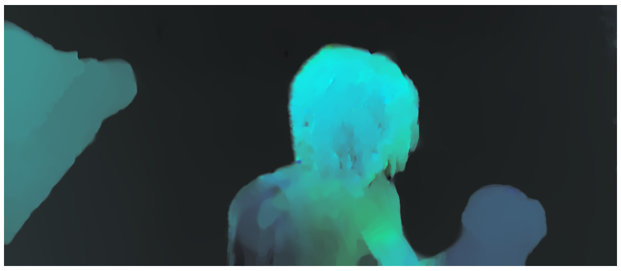
The sparse set of seeds (computed with Algorithm 4) is used as a reference to recover a dense flow field. This is done by iteratively growing the seeds by a local (patch-based) minimization of the functional in a proper order (see Section 2.1). This dense optical flow is then refined by a global minimization of the energy. The algorithm always works at the original image scale.

In order to describe the approach in more depth, we introduce some notation and assumptions.


(a) First frame and four initial seeds


(b) Color coding

(c) Second frame


(d) Our result

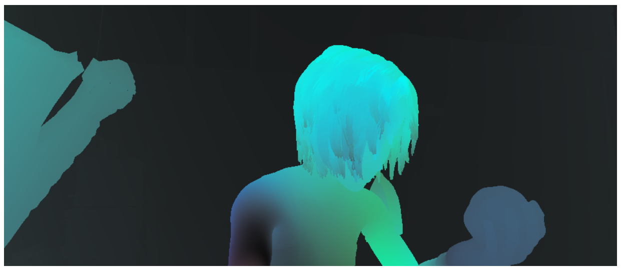

(e) Ground truth

Figure 1: Example illustrating that it is enough to have a single match per each region with an expected smooth motion; in this case, four seeds computed with SIFT (shown in red in (a)). The first row shows the original pair of frames I_t and I_{t+1}, where I_t has the seeds positions superimposed in red (head, arm, shoulder pad and background). Last row shows FALDOI's result and the ground truth. The energy functional is the classical TV_{ℓ_2}-$L1$ (denoted as $E^2 = E_D^2 + \beta E_R^2$ in Section 4.1).

Algorithm 1: `main algorithm` (end-to-end)

```
input  : Images $I_0, I_1$
input  : Functional $E$
input  : Image matcher $M$
input  : Patch size $w$
output : Final forward flow $\mathbf{u}^F$
$(\mathbf{x}^F, \mathbf{y}^F, \mathbf{x}^B, \mathbf{y}^B) \leftarrow$ compute-matches$(I_0, I_1, M)$            1. See Algorithm 2
$(Q^F, Q^B, \mathbf{u_o}^F, \mathbf{u_o}^B) \leftarrow$ sparse-flow$(\mathbf{x}^F, \mathbf{y}^F, \mathbf{x}^B, \mathbf{y}^F)$    2. See Algorithm 4
$(\mathbf{u}^F, \mathbf{u}^B) \leftarrow$ dense-flow$(Q^F, Q^B, \mathbf{u_o}^F, \mathbf{u_o}^B, E, w)$          3. See Algorithm 5
$(\mathbf{u}^F) \leftarrow$ flow-refinement$(I_0, I_1, \mathbf{u}^F, \mathbf{u}^B)$     4. Global minimization via Algorithm 8
```

Let us denote two consecutive image frames of a video sequence as $I_t, I_{t+1} : \Omega \to \mathbb{R}$, where Ω is a rectangle in $\mathbb{R}^2$. To compute the optical flow $\mathbf{u} : \Omega \to \mathbb{R}^2$ between I_t and I_{t+1}, we use a discrete set of matches $\mathcal{M} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$, $i = 1, \ldots, N$, and an energy functional $E(\mathbf{u})$, defined from I_t and I_{t+1}. Minimizing E on an appropriate set, a minimum $\mathbf{u}$ of E represents an optical flow between I_t and I_{t+1}. Section 4.1 describes several energies used in this paper. We assume that the discrete matches in $\mathcal{M}$ have been computed in such a way that $\mathbf{x}_i$ belongs to the image domain of I_t and $\mathbf{y}_i$ to the image domain of I_{t+1}. From these matches, we compute the initial set of seeds S by defining $\mathbf{u}(\mathbf{x}_i) = \mathbf{y}_i - \mathbf{x}_i$, $i = 1, \ldots, N$. Each seed $\mathbf{p}$ stores the corresponding pixel coordinates $\mathbf{x}^{\mathbf{p}}$, its optical flow $\mathbf{u}^{\mathbf{p}}$ and the local energy of the functional around it (see Algorithm 1).

2.1 Local Minimization

This step can be seen as an adaptive grouped coordinated descent approach guided by the lowest local values of the energy on patches centered at pixels. It is inspired from the match propagation principle [17], where a set of initial sparse matches is propagated to neighboring pixels using a region growing-like strategy. This principle was also used in the work of [14] to obtain a quasi-dense

Algorithm 2: `compute-matches` compute matches between I_0 and I_1

```
input  : Images $I_0, I_1$
input  : Image matcher $M$
input  : (optional) Threshold for DeepMatching matches $\rho$
output : Sparse flow
$(\mathbf{x}^F, \mathbf{y}^F) \leftarrow M(I_0, I_1)$                                   1a. Forward matches
$(\mathbf{x}^B, \mathbf{y}^B) \leftarrow M(I_1, I_0)$                                   1b. Backward matches
if $M$=DeepMatching then                         2. Filter matches by saliency (Algorithm 3)
    $(\mathbf{x}^F, \mathbf{y}^F) \leftarrow$ saliency-matches($I_0, I_1, (\mathbf{x}^F, \mathbf{y}^F), \rho$)      2a. forward matches)
    $(\mathbf{x}^B, \mathbf{y}^B) \leftarrow$ saliency-matches($I_1, I_0, (\mathbf{x}^B, \mathbf{y}^B), \rho$)      2b. backward matches
```

disparity map, assuming that the seeds and their neighboring pixels present similar disparity values. Moreover, it shares ideas with the coordinate descent methods, which are optimization techniques for multi-variable functions that are minimized by solving a sequence of one-variable minimization problems. Each problem improves an estimation by minimizing along a selected coordinate while all other remain fixed. Generally, each coordinate is visited several times to reach a minimum. The sweep pattern is the name used to define in which order the coordinates are visited. If the order is fixed, it is called path-wise coordinate descent [10] or cyclic coordinate.

In the case of FALDOI, the selection of the sweep pattern has a key role during the minimization process; it follows an adaptive choice of the coordinates driven by a seed growing algorithm which is based on the value of the local minimization of the energy in the patch centered at the coordinate. This process is managed by a priority queue where the potential candidates for each coordinate are inserted. Each of them has a related energy used to determine its position in the queue.

Next, we will first present the baseline algorithm for the local minimization step, where every pixel is visited just once, and then we will show how this is extended to several iterations that it also benefits from a further outlier filtering.

2.1.1 Baseline Algorithm: FALDOI

Initially, the seeds are inserted into the priority queue with zero energy. During the minimization process, new candidates are added to the queue. This collection of potential candidates for each coordinate is sorted based on their local energies.

Whenever an element is extracted from the queue, the energy E is minimized on a square patch centered on the element's pixel coordinates. Once this is done, we consider the pixel to be *fixed*. Next, the estimated optical flow values of its 4-connected neighbors (up, down, left and right) are inserted as potential candidates into the queue with the energy of the patch (after local minimization).

The aforementioned process is repeated until the priority queue is empty and a dense optical flow is obtained. There may be several candidates for the same pixel in the queue; when a candidate is extracted from the queue to fix it, if its corresponding pixel has already been fixed (by a candidate with lower energy) the candidate is discarded.

When the energy is minimized in a patch $\mathcal{P}$, we need to specify an initial flow for the unknown values in $\mathcal{P} \cap W$ (W is the set of locations where the optical flow has not been fixed). To do this we use the Laplace equation resulting from the minimization of the following Dirichlet energy

$$\min_{u_i} \quad \int_{\mathcal{P}} |\nabla u_i|^2 \, d\mathbf{x} \quad \text{s.t.} \quad u_i = u_i^0 \text{ in } \mathcal{P} \cap W^C,$$

where $i = 1, 2$, $\mathbf{u} = (u_1, u_2)$, and $\mathbf{u}^0 = (u_1^0, u_2^0)$ contains the optical flow of the fixed coordinates in $\mathcal{P} \cap W^C$ (W^C is the complementary set of W). The Euler-Lagrange equation derived from the previous energy is the Laplace equation, which is solved by gradient descent with Neumann boundary conditions.

The details of the minimization process are given in Appendix A and Algorithm 8. In practice, we consider patches of 11×11 pixels, if not stated otherwise. Bigger patches may be used but the computational cost increases linearly with their size. We perform 4 iterations of the minimization process, in every local patch (step 5 of Algorithm 6), of a version of $E(u)$ where the warped image has been linearized with one warping.

Algorithm 3: `saliency-matches` filter DeepMatching matches by saliency

```
input  : Images $I_A, I_B$
input  : Matches $\mathbf{m}$
input  : Threshold $\rho$
output : Filtered matches $\mathbf{m_f}$
$\mathbf{C_s} \leftarrow$ confidence-scores$(I_A, I_B, \mathbf{m})$          1. Compute confidence scores
$\mathbf{m_f} \leftarrow$ delete-outliers$(\mathbf{m}, \mathbf{C_s}, \rho)$          2. Delete outliers
$\mathbf{m_f} \leftarrow$ cut-deepmatches$(\mathbf{m_f})$          3. Cut filtered list (remove scores)
```

Algorithm 4: `sparse-flow` compute sparse flow from the matches

```
input  : Images $I_0, I_1$
input  : Matches $\mathbf{x}^F, \mathbf{y}^F, \mathbf{x}^B, \mathbf{y}^F$
output : Candidates queues $Q^F, Q^B$
$Q^F \leftarrow \emptyset, Q^B \leftarrow \emptyset$          1. Initialize empty priority queues
for $i = 1, \dots N$ do          2a. Add the forward matches with zero energy
    $Q^F.push(0, \mathbf{x}^F[i], \mathbf{y}^F[i] - \mathbf{x}^F[i])$          Initial forward flow: $\mathbf{u_o}^F[i] = \mathbf{y}^F[i] - \mathbf{x}^F[i]$
for $i = 1, \dots M$ do          2b. Add the backward matches with zero energy
    $Q^B.push(0, \mathbf{x}^B[i], \mathbf{y}^B[i] - \mathbf{x}^B[i])$          Initial backward flow: $\mathbf{u_o}^B[i] = \mathbf{y}[i]^B - \mathbf{x}[i]^B$
```

Algorithm 5: `dense-flow` compute dense flow from the initial sparse flow

```
input  : Images $I_0, I_1$
input  : Functional $E$
input  : Candidates queues: $Q^F, Q^B$
input  : Patch size $w$
output : Forward and backward flows $\mathbf{u}^F, \mathbf{u}^B$
$\mathbf{u}^F \leftarrow NULL$
$\mathbf{u}^B \leftarrow NULL$
$i \leftarrow MAX\_IT$
while $i > 0$ do          1. Compute local minimization (Algorithm 6)
    $\mathbf{u}^F \leftarrow$ basic-faldoi-growing$(I_0, I_1, \mathbf{u}^F, E, Q^F, w)$          1a. Forward
    $\mathbf{u}^B \leftarrow$ basic-faldoi-growing$(I_1, I_0, \mathbf{u}^B, E, Q^B, w)$          1b. Backward
    $(\mathbf{u}^F, \mathbf{u}^B) \leftarrow$ forwardBackward-pruning$(\mathbf{u}^F, \mathbf{u}^B)$          Algorithm 7
    $i \leftarrow i - 1$
```

Algorithm 6: `basic-faldoi-growing` (densify an incomplete flow)

```
Input  : Images $I_A, I_B$
Input  : Flow $\mathbf{u}_0$
Input  : Functional $E$
Input  : Priority queue $Q$
Input  : Patch size $w$
Output : Flow $\mathbf{u}$
while $Q.num\_elements() > 0$ do
    $e, \mathbf{x}, \mathbf{v} \leftarrow Q.pop()$                          1. get the candidate with lowest energy
    if $\mathbf{u}(\mathbf{x}) = NULL$ then
        $\mathbf{u}(\mathbf{x}) \leftarrow \mathbf{v}$                     2. fix the field for this candidate
    $\Omega_{\mathbf{y}} \leftarrow$ extract-patch($\Omega, \mathbf{y}, w$)                 3. extract patch of size $w \times w$
    $\mathbf{u} \leftarrow$ interpolate($\mathbf{u}, \Omega_{\mathbf{y}}$)                  4. fill-in missing values
    $\mathbf{u} \leftarrow$ flow-refinement($I_A, I_B, \Omega_{\mathbf{y}}, E, \mathbf{u}$)   5. minimize $E_{I_A,I_B}(\mathbf{u})$ over $\Omega_{\mathbf{y}}$ (Algorithm 8)
    $e \leftarrow E_{A,B}(\mathbf{u}, \Omega_{\mathbf{y}})$                   6. compute the energy of the solution
    for $\mathbf{y} \in \mathcal{N}(\mathbf{x})$ do                           7. add the neighbors of $\mathbf{x}$
        if $\mathbf{u}(\mathbf{y}) = NULL$ then
            $Q.push(e, \mathbf{y}, \mathbf{u}(\mathbf{y}))$                8. push this candidate value
```

Algorithm 7: `forwardBackward-pruning` consistency check

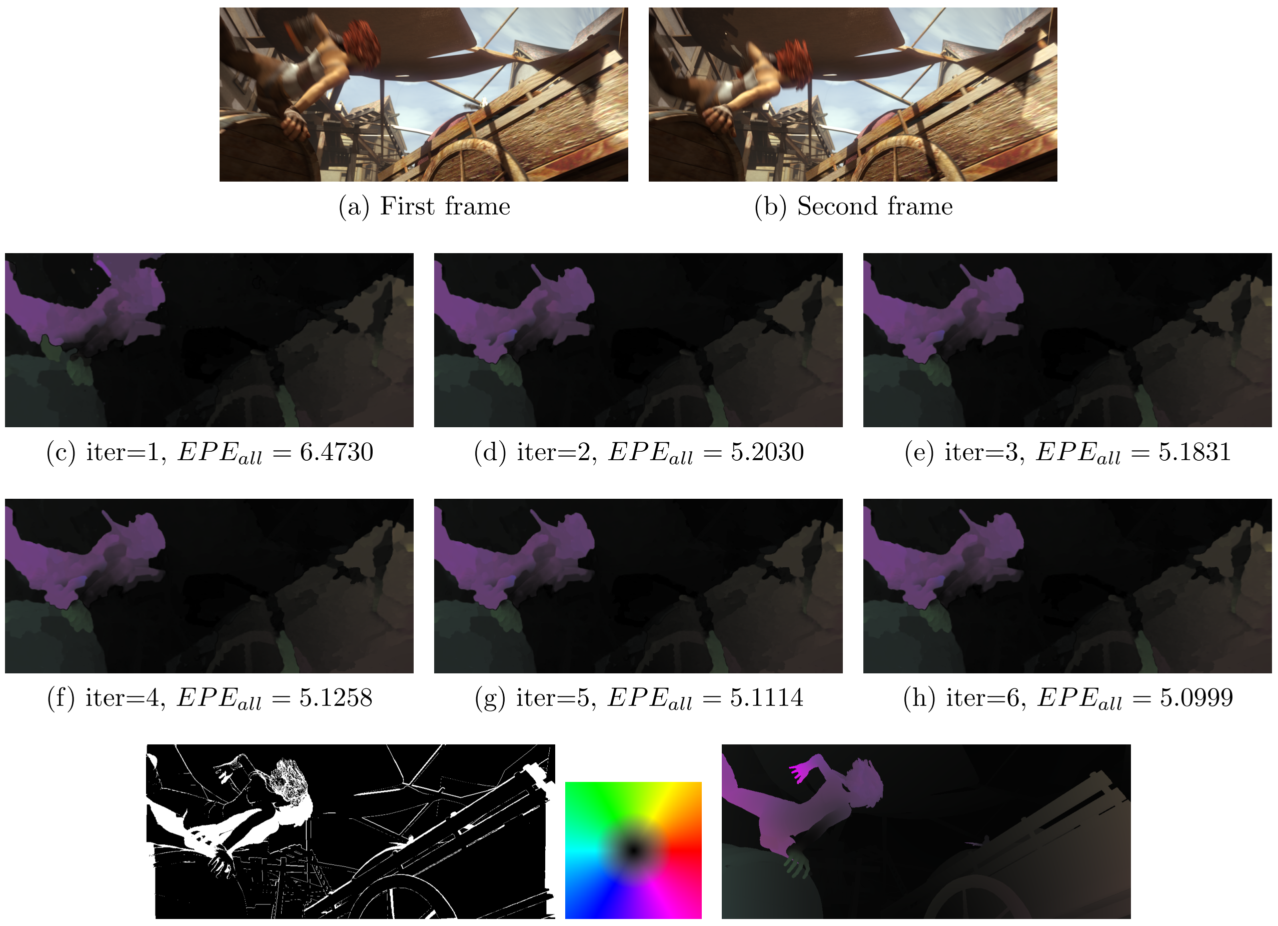
```
input  : Forward and backward flows $\mathbf{u}^F, \mathbf{u}^B$
input  : Image width and height: $w, h$
input  : Tolerance $\epsilon$
output : Updated forward and backward flows $\mathbf{u}^F, \mathbf{u}^B$
$size = w * h$
$n\_consistent = 0$
$consistent\_flow \leftarrow \vec{0}$
$\mathbf{u}_1^W \leftarrow NULL$                                    1a. Initialize warped flow (horizontal field)
$\mathbf{u}_2^W \leftarrow NULL$                                    1b. Initialize warped flow (vertical field)
bicubicInterpolation-warp($\mathbf{u}_1^B, \mathbf{u}_1^F, \mathbf{u}_2^F, \mathbf{u}_1^W, w, h$)   2a. Horizontal warping
bicubicInterpolation-warp($\mathbf{u}_2^B, \mathbf{u}_1^F, \mathbf{u}_2^F, \mathbf{u}_2^W, w, h$)   2b. Vertical warping
for $i = 0, \ldots size - 1$ do                                      3. Actual filtering
    $tolerance = || (\mathbf{u}_1^F[i] + \mathbf{u}_1^W[i], \mathbf{u}_2^F[i] + \mathbf{u}_2^W[i]) ||$
    if $tolerance > \epsilon$ then
        $consistent\_flow[i] = 0$                                   i-pixel flow is NOT consistent
    else
        $consistent\_flow[i] = 1$                                   i-pixel flow is consistent
        $n\_consistent++;$
```

2.1.2 Iterated FALDOI

We perform several iterations of the baseline Algorithm 6 due to the fact that in coordinate descent methods, each coordinate usually has to be visited several times to reach a minimum.

Moreover, between two consecutive iterations, we perform a pruning based on a forward-backward consistency check to remove wrong matches (Algorithm 7), specially in occluded areas and also due to self-similarity. The latter may appear by the presence of outliers in the initial seeds that have been expanded by the region growing. To achieve this, we compute both the forward, $\mathbf{u}^F$, and backward,

$\mathbf{u}^B$, optical flows between frames I_t and I_{t+1}. Then, the consistency check of these flows is calculated and the forward flow values at points $\mathbf{x} \in \Omega$ not verifying $\|\mathbf{u}^F(\mathbf{x}) + \mathbf{u}^B(\mathbf{x} + \mathbf{u}^F(\mathbf{x}))\| < \epsilon$ are removed; where $\epsilon > 0$ is a small constant (2 for all experiments performed in this paper). Similarly, the backward flow values at points $\mathbf{x} \in \Omega$ not verifying $\|\mathbf{u}^B(\mathbf{x}) + \mathbf{u}^F(\mathbf{x} + \mathbf{u}^B(\mathbf{x}))\| < \epsilon$ are removed.


(a) First frame (b) Second frame

(c) iter=1, $EPE_{all} = 6.4730$ (d) iter=2, $EPE_{all} = 5.2030$ (e) iter=3, $EPE_{all} = 5.1831$

(f) iter=4, $EPE_{all} = 5.1258$ (g) iter=5, $EPE_{all} = 5.1114$ (h) iter=6, $EPE_{all} = 5.0999$

(i) Occlusions mask (j) Ground truth

Figure 2: Iterated FALDOI strategy with the TV_{ℓ_2}-$L1$ energy and MAX_IT=6 in Algorithm 5. The evolution of the optical flow estimate is shown from left to right and top to bottom. The second and third row show the iterative local minimization (Section 2.1.2).

In the `iterated FALDOI (dense-flow)` algorithm, the iterations after the first one are slightly different from the baseline iteration of Section 2.1.1 as they start from a dense motion field with holes that arise after the forward-backward pruning instead of a sparse set of optical flow values. More precisely, there are two main differences with respect to the first iteration:

1. *Initialization of the queue.* The seeds that survive the forward-backward consistency check are inserted into a new queue with zero energy. The rest of optical flow values that survive are added with its associated energy.

2. *Initialization of the optical flow in a patch before local minimization.* Every time a local patch is minimized, an initial flow in the patch is needed. We make a distinction between the pixels that passed the forward-backward consistency test and those that did not. For the former, the initial flow is the most updated value from the last iteration for that pixel. For that, we store the flow value from the previous iteration in an auxiliary variable. On the other hand, for the non-surviving pixels, we fill-in the initial flow by using the Laplace interpolation explained above. This intra-patch initialization could also be used in the first baseline iteration of Section 2.1.1.

A pseudo-code of the `iterated FALDOI` strategy is presented in Algorithm 5. Figure 2 shows the relevance of several iterations in the local minimization step. The optical flow is improved from one iteration to the next (second and third rows). Specially, the motion boundaries are progressively better recovered and the effect of the piece-wise constant flow disappears. Furthermore, the `iterated FALDOI` strategy adds robustness against outliers in the initial seeds and against occlusions thanks to the forward-backward consistency check performed between two consecutive iterations. For instance, the densified flow obtained after the first iteration is incorrect on the leg that disappears behind the barrel. Nonetheless, this problem, which usually happens in the first iteration, is easy to detect and solve with the forward-backward pruning and the optical flow is improved in the subsequent iterations. After six iterations, the initial outlier regions are significantly reduced.

Also notice that the ground truth of the MPI-Sintel dataset is created from the *albedo* pass, where no external effects (motion blur, defocus blur, atmospheric effects or shading) are present. In this experiment the seeds have been computed with DeepMatching.

2.2 Global Minimization

The last step is a global minimization of $E(u)$, taking as initial condition the dense solution produced in the last step 2.1.2: the local minimization guided by the initial set of matches. Let us remark that we minimize the global energy at the finest scale, avoiding the multi-level approach. For both steps, local and global, the minimization uses the same energy functional with the numerical scheme detailed in Algorithm 8 and Appendix A. In this step, 5 warpings of the energy are performed.

Algorithm 8: `flow-refinement` minimization to compute refined optical flow

```
Input  : Two input frames $I_0, I_1$ and an initial optical flow $\mathbf{u}_0$
Output : Flow field $\mathbf{u}$
Initialize $\xi = \mathbf{v} = \vec{0}$;
Initialize $\mathbf{u} = \mathbf{u}_0$;
for $w \leftarrow 1$ to $N_{warps}$ do
    Compute $I_{t+1}(\mathbf{x}+\mathbf{v}_0(\mathbf{x}))$, $I^x_{t+1}(\mathbf{x}+\mathbf{v}_0(\mathbf{x}))$, $I^y_{t+1}(\mathbf{x}+\mathbf{v}_0(\mathbf{x}))$, using bicubic interpolation;
    while $n < N_{\max}$ or $tol < error$ do
        if $CSAD$ then
            Compute $\mathbf{v}$ as described in point 2.1 in Appendix A;
        else if $L1$ then
            Compute $\mathbf{v}$ as described in point 2.2 in Appendix A;
        if $NLTV$ then
            Compute $\xi$ or $p$ and $\mathbf{u}$ as described in point 1.1 in Appendix A;
        else if $TV_{\ell_2}$-$L1$ then
            Compute $\xi$ or $p$ and $\mathbf{u}$ as described in point 1.2 in Appendix A;
```

3 Code Optimization

In this section we detail the changes done to the code in order to significantly boost the algorithm's speed to create a live demo in the IPOL journal (under 1 minute and up to 1GB of memory) while keeping the same accuracy results of the original algorithm. Note that we have focused our efforts in optimizing the algorithm for all energy functionals but some of them are significantly slower than others (e.g.: $NLTV$-$CSAD$ versus classical TV_{ℓ_2}-$L1$, as denoted in the original paper).

As explained in Section 2, FALDOI performs two main steps: a local and a global minimization. The former can be parallelized by executing the forward and backward growings in separate CPUs since they do not share data. In particular, we need to add an OpenMP's parallel *for* instruction before a naive loop with two iterations, one per growing and OpenMP will handle the rest.

With the goal of using as many CPUs as possible, we choose to split the image in subimages (partitions) so that each of them can be processed by a CPU. More precisely, we divide both the forward (fwd) and backward (bwd) growing into $m \times n$ parts. We have to be careful selecting the number of partitions since there is a trade-off: a large number of partitions will led to more parallelization but it will also generate more discontinuities in the flow since each partition will have fewer seeds and the growing could suffer as a consequence; on the other hand, a smaller number of partitions may yield a negligible improvement in time since the memory overhead needed to copy the information for all threads may be too large. The decision will also be affected by the original image's size, since smaller images will probably benefit from fewer partitions. In order to avoid undesired flow discontinuities on the partition boundaries we divide the horizontal and vertical dimensions of the image into a different number of divisions ($m \times n$, where $m \neq n$) so that we can alternate between $m \times n$ and $n \times m$ partitions in consecutive iterations of the FALDOI algorithm.

We have tested two different types of partitions: 3×2 and 4×3, that is, 6 and 12 partitions per growing, respectively. This means that up to 6 or 12 CPUs (if available) may be working at once instead of only 2 (fwd + bwd). It is also important to note that in order to let the initial seeds grow significantly, the first iteration is performed using the whole image, without partitions. This ensures that latter iterations work with a good initial flow estimate but it also acts as a bottleneck for the algorithm's performance (as only the forward and backward growings for the whole image can be done in parallel, i.e.: 2 cpus). One can clearly notice that in the in-depth analysis presented in Tables 1, 2, 3, 4, 5 and 6.

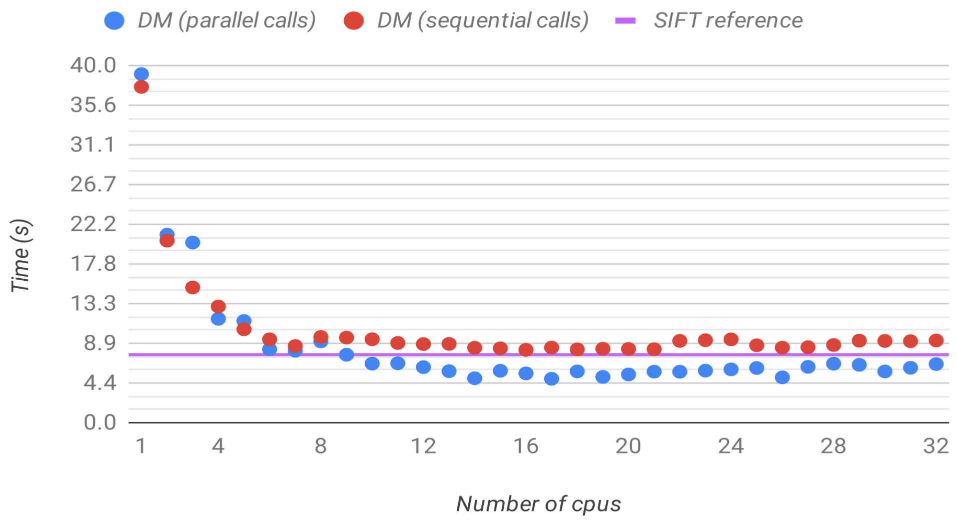


Figure 3: Comparison of execution times of sequential and parallel calls of Deepmatching vs the number of CPUs used. In red, forward and backward matches computed sequentially, in blue both calls in parallel. SIFT reference in purple (single-threaded).

We have also parallelized the global step by using detailed profiling to find the critical sections in the code that take most of the time. We have sorted them, from more to less critical. Then, we have optimized their internal for loops one at a time, measuring the improvement (in percentage) and only keeping the ones that have given a significant improvement. Finally, the chosen ones can be combined to reduce the execution time further.

As for the matches, we computed both SIFT and DeepMatching in parallel (for frames I_t and I_{t+1}). We used the *multiprocessing*[2] library in Python to call the binaries for computing matches

[2]Multiprocessing library for Python3: `docs.python.org/3/library/multiprocessing.html`

(SIFT or DeepMatching) at once in order to compute the forward and backward matches in parallel and thus halve the computation time compared to the sequential execution. Moreover, the SIFT implementation does not exploit multithreading so its execution time remains fairly constant for any number of CPUs. On the other hand, the implementation of DeepMatching is fully multi-threaded and significantly reduces its execution time as the number of available CPUs increases (see Figure 3). Note that the reported times have been computed for only one image in the MPI-Sintel Final pass (clean pass may take longer as more matches are found, specially for SIFT due to the lack of multi-threading). We could obtain a similar boost in speed for SIFT by using other implementations which are more optimized such as that integrated in the VLFeat[3] open source library by A. Vedaldi and B. Fulkerson.

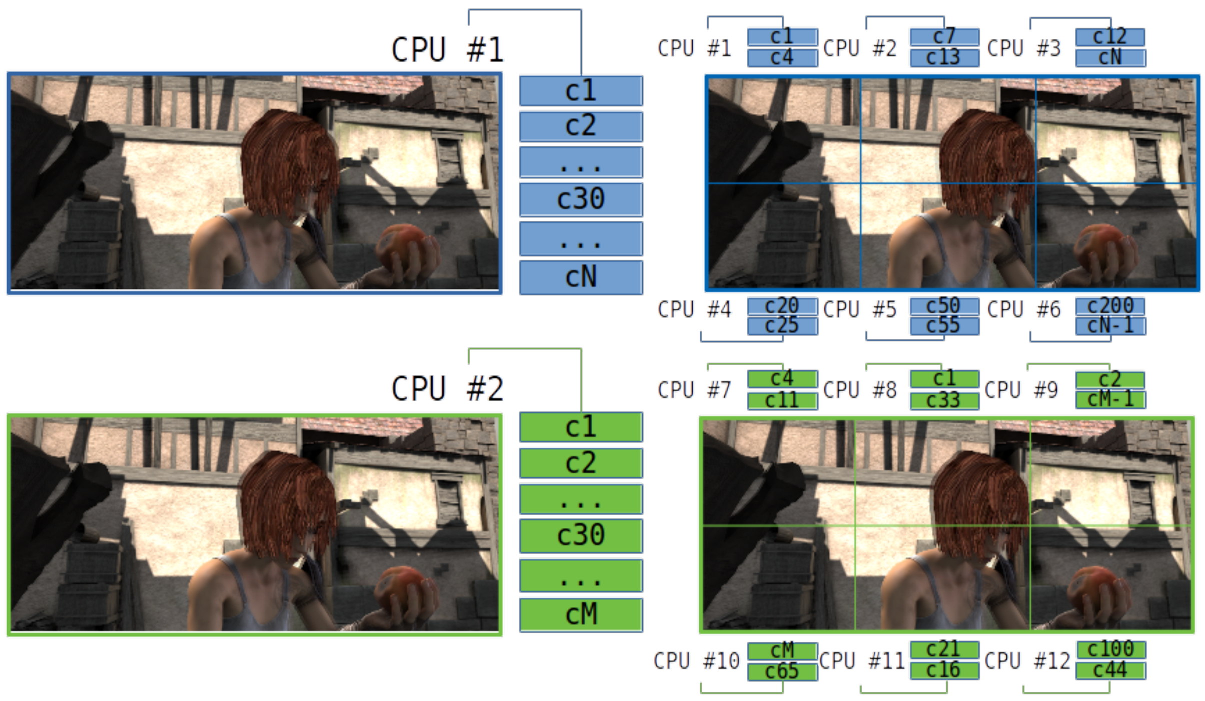


Figure 4: Illustration of the CPUs usage in the original code (left) and the optimized one (right). Note how, thanks to the use of partitions, several growings can be processed at once instead of sequentially.

Figure 4 illustrates the use of CPUs in the original and the speedup code: in the former, only the forward and backward growings for the whole image were being executed at the same time while in the latter, up to $2 \times m \times n$ growings can be executed at once. Each growing handles its own priority queue that, as explained in Section 2.1.1, stores the candidates to be visited during the local minimization. It is important to notice that these queues are significantly smaller than the original queue (one per image) and hence faster to manage. This can be clearly seen if one compares the case for one CPU with and without partitions: despite both cases use a single CPU, handling $2m \times n$ smaller queues is faster than doing the same with 2 significantly bigger ones (see for example, Tables 1 and 2).

Another important aspect to remark, is that the performance of parallelization can be reduced significantly in two critical scenarios. First, when we have few matches/seeds; for instance, when we employ SIFT with a texture-less image, which yields very few seeds that will probably fall only in a few partitions (and thus the remaining partitions will not contain seeds). As a result, these empty partitions will have empty queues and will not be used at all, reducing the number of active CPUs and thus the general code speedup. To simplify the implementation of such cases, we revert to using

[3]VLFeat: An Open and Portable Library of Computer Vision Algorithms: www.vlfeat.org/

the whole image (without partitions) when any of the queues is empty (notice that, on the other hand, the forward and backward optical flow estimations are always computed in parallel). Secondly, we may have the opposite case: a lot of matches, which also increases the execution time, yielding bigger queues that are harder to handle and hence, slower. It is important to notice that in such cases, the following iteration is divided into partitions again as long as no queue is still emtpy.

In these critical cases, the parallelized code with partitions may take almost the same time as its counterpart without partitions and may be above the 1 minute threshold. If that is the case, your execution may be stopped by the server. If so, we advise you to do one of the following things: if you are using SIFT, try DeepMatching instead as it gives more initial seeds and is less likely to yield as many empty partitions as SIFT (depends on the number of outliers); alternatively, execute the code locally with the original parameters but be aware that execution time will be notably increased (also depending on your hardware).

In order to properly evaluate the speedup of the algorithm, we have averaged the timings reported by the algorithm with 17 images that include the critical cases commented above so the estimation is more accurate. It is important to notice that including such cases, the average execution time grows slightly with respect to not considering these cases. Nevertheless, these critical cases are very rare even in the challenging final pass of MPI-Sintel (under a 1%).

We opt to use 3×2 instead of 4×3 partitions because the difference in execution time is negligible. This is probably due to the bigger memory overhead of using more threads in parallel. This is why all results in this paper with the TV_{ℓ_2}-$L1$ functional have been computed with a 3×2 partition.

We also report the average endpoint error (EPE) for each setup to proof that the improvement in speed has a negligible effect on the accuracy of the method. It is defined as follows

$$EPE = EPE_{all} = \frac{1}{n}\sum_{i\in\Omega}\sqrt{(u^i_{es}-u^i_{gt})^2+(v^i_{es}-v^i_{gt})^2}, \tag{1}$$

where n is the number of pixels, Ω is the image domain. The ground truth optical flow is composed of the vectors (u_{gt}, v_{gt}) and similarly, the estimated flow is defined by the fields (u_{es}, v_{es}).

Step	NP1	WP1	NP2	WP2	NP4	WP4	NP8	WP8	NP16	WP16
Total	216.98	183.08	130.74	109.27	129.39	84.37	123.64	74.62	123.49	**66.55**
Descriptors	8.18	8.27	7.31	4.17	4.22	4.17	4.15	4.09	4.14	4.11
Matches	7.80	7.63	6.18	3.86	3.92	3.84	3.80	3.78	3.60	3.80
Local min.	178.28	158.60	108.02	86.70	110.20	66.02	106.09	58.18	106.51	50.73
LM-iter1	52.62	53.08	46.57	28.08	28.31	27.81	27.13	26.99	27.23	27.11
LM-iter2	49.86	42.19	26.86	22.81	27.48	15.63	25.93	13.26	26.05	10.14
LM-iter3	49.64	43.25	27.12	22.56	27.96	13.48	26.11	10.81	26.38	6.63
LM-last	24.97	20.85	24.63	11.09	25.27	8.24	25.54	5.81	25.46	5.27
Global min.	22.61	20.30	14.65	14.49	10.42	10.22	9.53	8.52	8.91	7.85
EPE	22.13	22.19	22.13	22.19	22.13	22.19	22.13	22.19	22.13	22.19
Speedup vs OG	4.91	5.82	4.38	5.24	4.23	6.48	6.51	10.79	5.81	10.78
Speedup vs NP1	N/A	1.2	1.7	2.0	1.7	2.6	1.8	2.9	1.8	3.3

Table 1: FALDOI's mean execution time (seconds) for 11×11 patch size with SIFT matches. The results with (WP) and without partitions (NP) are compared for different number of CPUs (number next to initials). OG denotes FALDOI's original (non-optimized) code. The energy functional is the TV_{ℓ_2}-$L1$.

Tables 1 and 2 show a detailed profiling of FALDOI for different number of CPUs and default parameters (see Appendix B). We have included the total execution time and the partial timings from the matching computation, each local growing iteration and the global estimation. Note that the 'Total' time is composed of the addition of 'Descriptors' (only for SIFT), 'Matches', 'Local min.' and

Step	NP1	WP1	NP2	WP2	NP4	WP4	NP8	WP8	NP16	WP16
Total	239.93	216.76	159.33	135.42	133.00	83.43	126.15	71.00	122.18	**58.13**
Matches	37.11	37.62	37.05	36.31	11.46	11.40	8.02	8.53	5.24	5.13
Local min.	180.19	158.53	107.72	83.87	110.31	61.62	108.31	53.49	107.85	44.87
LM-iter1	53.01	53.21	27.64	28.41	28.20	27.62	27.64	27.03	27.39	27.27
LM-iter2	50.70	42.85	27.06	22.39	27.88	13.24	27.00	10.81	26.89	6.62
LM-iter3	50.04	43.44	26.87	22.39	27.56	12.54	26.93	9.68	26.67	5.30
LM-last	25.05	20.18	24.81	10.43	25.21	7.25	25.36	4.50	25.47	3.97
Global min.	21.68	20.24	14.22	13.58	10.89	10.07	9.48	8.64	8.90	7.78
EPE	17.89	16.97	17.89	16.97	17.89	16.97	17.89	16.97	17.89	16.97
Speedup vs OG	5.97	6.60	6.87	8.09	8.11	12.94	10.51	18.68	10.30	21.65
Speedup vs NP1	N/A	1.1	1.5	1.8	1.8	2.9	1.9	3.4	2.0	4.1

Table 2: Execution times with 11×11 patch size and DeepMatching (see Table 1 for details).

Step	NP1	WP1	NP2	WP2	NP4	WP4	NP8	WP8	NP16	WP16
Total	217.35	147.16	101.31	86.31	97.49	71.10	92.25	62.62	91.99	**54.10**
Descriptors	8.24	8.30	4.27	4.42	4.23	4.47	4.14	4.23	4.12	4.21
Matches	7.68	7.69	3.99	3.99	3.95	3.93	3.81	3.81	3.63	3.87
Local min.	179.41	110.98	78.27	63.15	78.25	51.60	74.88	45.21	75.48	39.59
LM-iter1	53.24	37.43	20.25	20.19	20.04	20.19	19.23	19.30	19.25	19.73
LM-iter2	49.65	29.75	19.41	16.26	19.48	12.31	18.11	10.43	18.36	8.43
LM-iter3	49.93	29.88	19.59	16.15	19.64	10.23	18.30	8.08	18.64	4.80
LM-last	25.20	14.31	17.62	8.41	17.71	6.58	17.84	5.11	17.81	4.46
Global min.	21.99	20.14	14.72	14.69	11.00	10.98	9.36	9.31	8.48	8.60
EPE	23.36	23.25	23.36	23.25	23.36	23.25	23.36	23.25	23.36	23.25
Speedup vs OG	4.26	6.29	5.62	6.60	5.61	7.69	8.00	11.79	7.80	13.26
Speedup vs NP1	N/A	1.5	2.1	2.5	2.2	3.1	2.4	3.5	2.4	4.0

Table 3: Execution times (similar to Table 1) with 9×9 patch size and SIFT matches.

Step	NP1	WP1	NP2	WP2	NP4	WP4	NP8	WP8	NP16	WP16
Total	188.45	172.12	131.45	114.46	99.68	72.07	94.33	59.35	89.83	**47.83**
Matches	37.71	37.33	37.61	37.55	11.34	12.71	8.83	9.54	5.07	5.49
Local min.	128.52	78.81	77.15	75.82	75.92	106.39	61.92	48.09	40.45	33.43
LM-iter1	37.56	20.19	19.58	19.25	19.15	37.45	20.21	20.32	19.29	19.55
LM-iter2	36.12	19.79	19.23	18.52	19.08	29.88	15.99	10.26	8.02	4.84
LM-iter3	35.66	19.70	19.29	18.41	18.60	29.67	16.14	9.76	7.40	3.91
LM-last	17.79	17.76	17.62	18.27	17.73	13.91	7.35	5.33	3.40	2.88
Global min.	21.86	14.67	10.86	9.33	8.49	21.57	14.64	10.93	9.01	8.54
EPE	17.86	17.33	17.86	17.33	17.86	17.33	17.86	17.33	17.86	17.33
Speedup vs OG	6.57	9.42	9.66	10.22	10.56	5.51	10.02	15.92	18.70	23.21
Speedup vs NP1	N/A	1.1	1.4	1.6	1.9	2.6	2.0	3.2	2.1	3.9

Table 4: Execution times (similar to Table 2) with a 9×9 patch and DeepMatching.

'Global min.'. Additionally, the local minimization is the sum of all its iterations, that is, 'LM-iter1' through 'LM-last'.

We have also included an extra row for the speedup factor between that particular setup and the original (non-optimized) FALDOI code executed under the same conditions.

We have also included results for two other patch sizes (9×9 pixel size in Tables 3 and 4, and 7×7 in Tables 5 and 6) detailing the differences between the original code, the current code without

Step	NP1	WP1	NP2	WP2	NP4	WP4	NP8	WP8	NP16	WP16
Total	119.87	106.58	73.85	62.86	70.07	52.50	65.94	46.00	66.12	**42.61**
Descriptors	8.29	8.23	4.29	4.26	4.26	4.24	4.15	4.15	4.17	4.11
Matches	7.64	7.59	3.74	4.02	3.96	3.99	3.76	3.78	3.91	3.82
Local min.	82.12	71.97	50.83	40.80	50.83	31.96	48.72	29.53	49.45	26.65
LM-iter1	23.94	23.81	12.92	12.92	12.78	12.75	12.20	12.20	12.30	12.31
LM-iter2	22.63	18.73	12.54	10.46	12.58	7.74	11.63	6.73	12.18	5.62
LM-iter3	22.79	18.75	12.68	10.37	12.73	6.79	11.72	5.74	12.24	4.12
LM-last	11.33	8.79	11.32	4.90	11.35	3.68	11.78	2.63	11.31	2.35
Global min.	21.78	20.03	14.70	13.73	10.97	10.27	9.25	8.49	8.50	7.95
EPE	22.88	22.89	22.88	22.89	22.88	22.89	22.88	22.89	22.88	22.89
Speedup vs OG	7.65	8.60	7.73	9.08	7.81	10.43	12.17	17.45	10.88	16.88
Speedup vs NP1	N/A	1.1	1.6	1.9	1.7	2.3	1.8	2.6	1.8	2.8

Table 5: Execution times (similar to Table 1) with a 7×7 patch and SIFT matches.

Time events	NP1	WP1	NP2	WP2	NP4	WP4	NP8	WP8	NP16	WP16
Total	141.82	128.79	103.58	87.90	72.78	52.05	67.62	43.60	63.51	**35.91**
Matches	37.72	37.71	37.64	37.90	11.50	11.51	9.04	8.32	5.06	5.21
Local min.	82.05	70.16	50.92	39.00	50.09	29.86	48.84	25.87	49.60	22.36
LM-iter1	23.66	23.81	12.83	12.85	12.54	12.74	12.17	12.18	12.27	12.23
LM-iter2	22.89	18.61	12.67	10.13	12.51	6.28	11.76	4.99	12.38	3.43
LM-iter3	22.75	18.47	12.65	10.18	12.51	6.03	11.68	4.83	12.20	2.91
LM-last	11.33	8.53	11.39	4.57	11.19	3.20	11.80	2.03	11.39	1.82
Global min.	21.69	20.56	14.60	13.84	10.86	10.34	9.39	9.07	8.50	7.99
EPE	17.85	17.41	17.85	17.41	17.85	17.41	17.85	17.41	17.85	17.41
Speedup vs OG	7.31	8.05	8.24	9.71	11.56	16.17	14.75	22.87	15.65	27.67
Speedup vs NP1	N/A	1.1	1.4	1.6	1.9	2.7	2.1	3.3	2.2	3.9

Table 6: Execution times (similar to Table 2) with a 7×7 patch and DeepMatching.

win_size	11×11	9×9	7×7
w.r.t. to current code w. downscale $= 1$			
Increase in error	5.13%	3.9%	5.6%
Decrease in matching time	8.7x	8.7x	8.7x
Decrease in total time	1.7x	1.9x	2.2x
Decrease in memory used	8x	8x	8x
w.r.t. to original code w. downscale $= 1$			
Increase in error	−0.4%	2.88%	−2.70%
Decrease in matching time	104.9x	104.9x	104.9x
Decrease in total time	18.6x	19.8x	23.4x
Decrease in memory used	8x	8x	8x

Table 7: Advantages and disadvantages of using a $downsample = 2$ for deepmatching (half-resolution) instead of the original scale.

partitions and its version with partitions. One can see that for 11×11 and SIFT matches (Table 1), the execution time is slightly above 60 seconds due to the inclusion of a very complex example with very few seeds in the processed images. It is also important to note the speedup introduced by using partitions even when only one CPU is available: it significantly reduces the execution time. One can also see that, despite being quite slower with few CPUs, DeepMatching ends up being faster with

16 CPUs, resulting in an overall smaller execution time thanks to its better scaling (Figure 3). In that regard, note that the DeepMatching version already has an execution time below the 1 minute threshold while the SIFT version is still above it.

Comparing it to FALDOI's original source code, we can see that the speedup factor is always above 4 and it is even larger for more CPUs which is mainly due to the fact that the original code did not scale properly when using 8 or more CPUs. This may be due to the fact that FALDOI's original code parallelizes several *for-loops* which are nested inside others which yields a big memory overhead and takes more time. Additionally, in the case where DeepMatching seeds are used, the scalability commented above is not reproduced since the original code used an older version of DeepMatching that only used 1 CPU. This fact is critical as one can see in Table 7, where the combination of a multi-threaded version of DeepMatching and parallel calls for the forward and backward image yields a 8.7 times faster matching computation. Moreover, FALDOI' original code computed the seeds at the original image scale instead of at half-resolution, as the current method does, which produces reasonably similar results (about 5% worse for an 11×11 patch) but it is almost 2x faster in mean execution time (averaged over several frames) than using the original scale. The difference is more pronounced when we use the original image scale (more computationally expensive) for which the current version of the code is 18.6 times faster in execution time due to the null scaling of DeepMatching and the other speedup optimizations commented throughout this section. This time reduction is consistent with the other two patch sizes tested. Another reason for not using $downscale = 1$ is that it uses more memory than the limit specified by IPOL for the online demo. All remaining tests, if not stated otherwise, use half-resolution images to compute the DeepMatching seeds.

In the second pair of tables 3, 4 (9×9 patch), we can see the same trend but with all timings reduced approximately by a 25% with respect to the default parameters and SIFT as the matching algorithm. The difference between SIFT and DeepMatching is maintained and the minimum run-time reported is below the threshold for both matchers. Finally, in the third pair of tables 5, 6 (7×7 patch size), the same observations can be drawn, obtaining a speedup of about 60% with respect to the default setup. The minimum time is around 35 seconds per image (with resolution around 1024×436). Nevertheless, in this last scenario the error is bigger than in the default case despite being approximately equal to the one reported for a 9×9 patch size.

In Section 4.1, several possibilities for the optical flow energy functional are presented, all of them available in the provided code. In particular TV_{ℓ_2}-$L1$, TV_{ℓ_2}-$CSAD$, $NLTV$-$CSAD$, and $NLTV$-$L1$ models. The fastest functional is TV_{ℓ_2}-$L1$ while the slowest one is the $NLTV$-$CSAD$. Given the nature of $NLTV$, a GPU implementation would probably boost its speed remarkably.

This is the reason why in the live demo in IPOL, only the TV_{ℓ_2}-$L1$ functional will be available. Nonetheless, the provided code can be executed in local mode with all energies mentioned in Section 4.1.

In all cases, the speedup with respect to the original code is very noticeable, specially in the case of TV_{ℓ_2}-$L1$.

4 Results

FALDOI assumes that two ingredients are given: a discrete set of correspondences between two frames of a video (seeds) and an optical flow energy functional, namely, $E(\mathbf{u})$. Section 4.1 presents the different energies we have used in this paper and Section 4.2 briefly discusses several possibilities for the initial sparse correspondences. Later on, in Section 4.3 we provide a quantitative and qualitative comparison among the several possibilities for the energy terms, as well as a comparison between FALDOI and several state-of-the-art methods, including methods based on the combination of sparse matches and variational techniques [5, 16, 24, 31] or methods that rely on CNNs [9, 13, 29] to estimate the flow.

4.1 Different Possibilities for the Energy

The proposed framework is independent from the energy functional $E(\mathbf{u})$ and we validate it by using several energies. Following most of the optical flow variational approaches in the literature, the different possibilities will share the common feature of being made of two terms: a data fidelity term $E_D(\mathbf{u})$ and a regularization term $E_R(\mathbf{u})$,

$$E(\mathbf{u}) = E_D(\mathbf{u}) + \beta E_R(\mathbf{u}), \tag{2}$$

where $\mathbf{u} : \Omega \to \mathbb{R}^2$, Ω is the image domain, $I_t, I_{t+1} : \Omega \to \mathbb{R}^d$ are two consecutive frames ($d = 1$ for gray level images and $d = 3$ for color images), and $\mathbf{u} = (u_1, u_2)$ represents the motion field between them.

Let us also state that FALDOI allows the inclusion of other terms, e.g.: a third term dealing with the occlusions, as in [4, 1]. In fact, the provided code includes an energy for the joint estimation of optical flow and occlusions for the $TV_{\ell_2} - L1$ functional [22] as proposed in [4]. We do not describe it in this document or used it in the online demo but we have opted to leave it in the code for the sake of completeness. If you wish to test the code with occlusions, please go to the Github repository linked at the top of this article.

In order to have a robust data cost, specially under illumination changes, we use a convex and continuous approximation of the data cost based on the Census transform [12, 27, 34], approximated by a sum of centralized absolute differences, denoted as $CSAD$ [30], and defined as

$$E_D^1(\mathbf{u}) = \int_\Omega \int_\Omega |I_t(\mathbf{x}) - I_t(\mathbf{y}) - I_{t+1}(\mathbf{x}+\mathbf{u}) + I_{t+1}(\mathbf{y}+\mathbf{u})| \chi(\mathbf{x}-\mathbf{y}) d\mathbf{y} d\mathbf{x}, \tag{3}$$

where χ denotes the characteristic function of a square of size $P \times P$ centered at the origin ($P = 7$ in our experiments, as in [30]).

Secondly, we also consider the classical point-wise L^1 data term that imposes the well-known brightness constancy assumption, namely,

$$E_D^2(\mathbf{u}) = \int_\Omega |I_{t+1}(\mathbf{x}+\mathbf{u}) - I_t(\mathbf{x})| \, d\mathbf{x}. \tag{4}$$

It is important to correctly preserve the motion boundaries in order to obtain an accurate optical flow. These boundaries are usually aligned with image boundaries, which has motivated the use of edge detectors (e.g.: [8, 15]) in previous optical flow methods [16, 20, 24]. Instead, FALDOI opts for using the Non-Local Total Variation ($NLTV$) regularizer. $NLTV$ was used for optical flow estimation in [32], where the authors show its ability to better recover motion boundaries, particularly in low-textured areas, occluded regions, and in small objects (when used in a coarse-to-fine scheme). In our case, a non-local regularizer that better captures motion discontinuities is also very useful in the local minimization step where the initial correspondences are densified. The regularizer is applied to each flow channel independently

$$E_R^1(\mathbf{u}) = \int_\Omega \int_\Omega \omega(\mathbf{x}, \mathbf{y}) \left(|u_1(\mathbf{x}) - u_1(\mathbf{y})| + |u_2(\mathbf{x}) - u_2(\mathbf{y})| \right) d\mathbf{y} d\mathbf{x}, \tag{5}$$

where $\omega(\mathbf{x}, \mathbf{y}) = \frac{1}{\mathcal{W}(\mathbf{x})} e^{\frac{-\Delta_c(\mathbf{x},\mathbf{y})}{\sigma_c}} e^{\frac{-\Delta_s(\mathbf{x},\mathbf{y})}{\sigma_s}}$, $\Delta_c(\mathbf{x}, \mathbf{y})$ denotes the Euclidean distance between the color values at $\mathbf{x}$ and $\mathbf{y}$ in the Lab space, $\Delta_s(\mathbf{x}, \mathbf{y})$ is the Euclidean distance between points $\mathbf{x}$ and $\mathbf{y}$,

$\mathcal{W}(\mathbf{x}) = \int_\Omega \omega(\mathbf{x}, \mathbf{y}) d\mathbf{y}$ is a normalizing constant, and σ_c, $\sigma_s > 0$ are constant parameters (set to $\sigma_c = 2$ and $\sigma_s = 2$).

As before, with the purpose of testing the proposed minimization algorithm with different energies, we consider other regularization terms such as the classical coupled Total Variation (called $TV_{\ell_2}(u)$ in [28]), that is,

$$E_R^2(\mathbf{u}) = \int_\Omega \|\nabla \mathbf{u}\|_2 d\mathbf{x} = \int_\Omega \sqrt{|\nabla u_1(\mathbf{x})|^2 + |\nabla u_2(\mathbf{x})|^2} d\mathbf{x}, \tag{6}$$

Appendix A details the optimization algorithms for all the energy terms. Let us say that the data and the regularization terms are decoupled and standard methods such as primal-dual, thresholding or median schemes are used.

4.2 Different Possibilities for the Initial Set of Seeds

In this work, the initial seeds are computed with one of the sparse matchers in the literature. There are several methods that provide a set of sparse matches between two different images containing common objects, representing two views of a scene or, as in our case, two frames of a video. Some of them are based in the estimation of distinctive point location and matching [19, 21]. Being based on local properties, they can be used to estimate arbitrarily large displacements. In our experiments, we use SIFT [19] or DeepMatching [31]. Although SIFT is very effective, it has some disadvantages when dealing with small texture-less objects or with non-rigid deformations. The DeepMatching algorithm handles these problems and generates a more dense set of matchings. We have compared these two different methods to compute the initial set of matches (the seeds). The first column of Figures 5 and 6 show some examples of the set of seeds obtained by each one of the algorithms.

We use the SIFT [25] and DeepMatching[4] implementations, with its default parameters.

With the aim of avoiding outliers in homogeneous areas, when we employ DeepMatching, we initially remove the seeds having a low local saliency, which is determined by the minimum eigenvalue of the autocorrelation matrix computed locally (Algorithm 3). If the value is below a threshold (set to 0.045 in our experiments), these seeds are removed. This pruning is also used in other works, e.g.: [24].

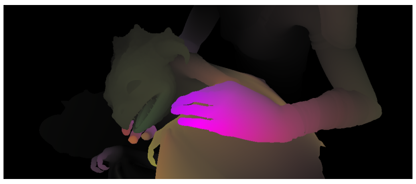
In FALDOI, the authors claim that the algorithm is able to recover the dense motion even if very few initial seeds are available; the only condition is to have at least one correct seed in every region $\Omega_i \subset \Omega$ where the motion is smooth. As a proof of concept, in Figure 1 we use a single seed per Ω_i, which has been extracted from the ground truth flow. The estimated optical flow compares favorably to the ground truth even in the cases where the initial set of matches is extremely sparse. In contrast, other sparse-to-dense methods, like EpicFlow, with very competing results are not capable of estimating the optical flow in this challenging situation (see Figure 9 for a more complex example of this). Another example is shown in Figure 5 where similar optical flow results are obtained independently of the cardinality of the set of seeds, which have been computed using either SIFT (Figure 5, second row) or DeepMatching (Figure 5, third row) algorithms. Let us notice that DeepMatching produces in general more matches than SIFT but also introduces potentially more outliers.

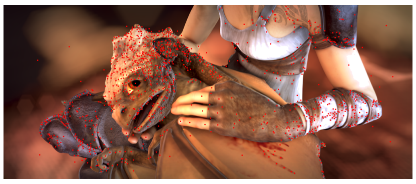
On the other hand, Figure 6 shows a situation where the lack of seeds in some regions Ω_i (second row, seeds provided by SIFT matches) produces an incorrect flow estimation in these areas. A much better estimation is obtained when there is at least one seed in these regions, as it happens when we use as seeds the matches provided by DeepMatching (Figure 6, third row).

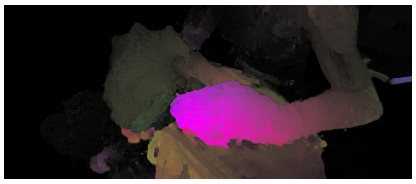
[4]Deep Convolutional Matching: `http://lear.inrialpes.fr/src/deepmatching/`

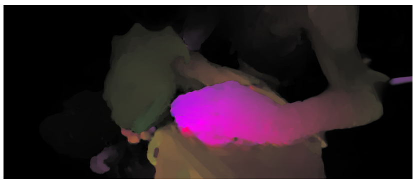
(a) First frame

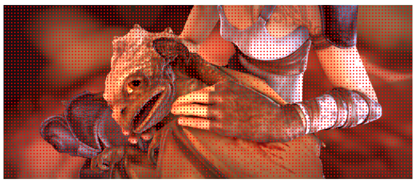
(b) Second frame

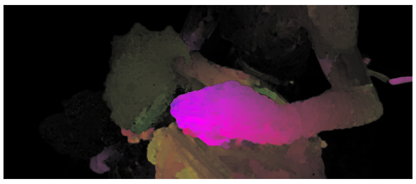

(c) Ground truth


(d) SIFT seeds


(e) Local


(f) Global


(g) DeepMatching seeds


(h) Local

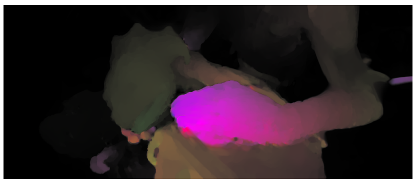
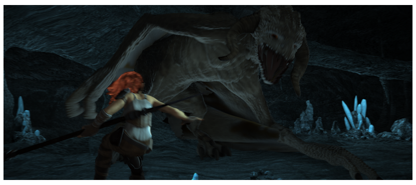
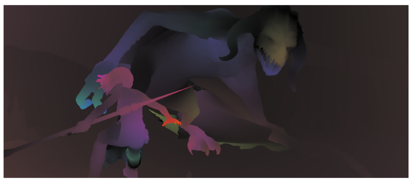
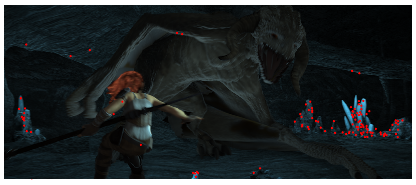

(i) Global

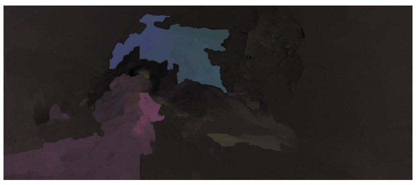
Figure 5: Sequences with enough initial seeds for both matchers to recover a correct dense flow. The energy functional used is the classical TV_{ℓ_2}-$L1$.

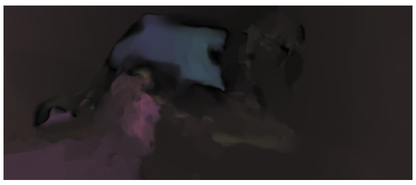

(a) First frame

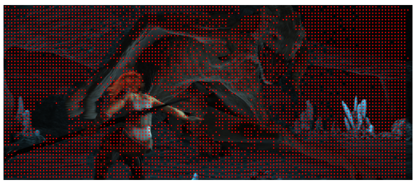
(b) Second frame

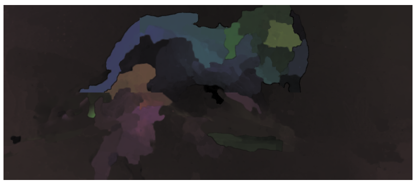

(c) Ground truth


(d) SIFT seeds


(e) Local


(f) Global


(g) DeepMatching seeds


(h) Local

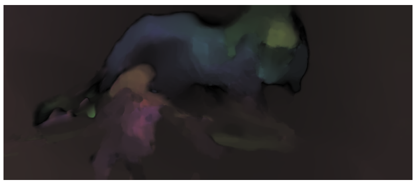

(i) Global

Figure 6: Sequence where only the second matcher (DeepMatching [31]) gets enough seeds to estimate a correct flow. The energy functional is the classical TV_{ℓ_2}-$L1$.

4.3 Experimental Results

The method has been tested on two publicly available databases: Middlebury [3] and MPI-Sintel [6], and on proof of concept examples, chosen in order to obtain a better understanding of the key features of the algorithm. Let us remark that all results have been obtained by using the grayscale versions of the original color frames. The color version is only used to compute the seeds in case they come from the DeepMatching algorithm and to compute the local regularization weights in case of the Non-Local Total Variation as regularization term. All the results in this section have been obtained with the FALDOI minimization strategy (Algorithm 1) presented in Section 2.1.2. If not stated otherwise, the parameters used are the same for all experiments, listed in Appendix B.

First, we present a comparison of FALDOI against the multi-scale approach using different functionals. Our implementation of those multi-scale approaches is based on the implementation published in [26]. In order to assure that the method is a real alternative to the coarse-to-fine warping strategy and, therefore, also valid for sequences that do not have large displacements of small objects, experiments on the Middlebury optical flow dataset [3] have been performed for both approaches. Table 8 shows how FALDOI achieves better results in all the samples in the dataset[5], even if the difference is slight is some cases. Nevertheless, it is interesting to note that the functional that obtains the better results changes for some of the sequences. For instance, the sequence *urban3* yields much worse results for the TV_{ℓ_2}-$CSAD$ and the $NLTV$-$CSAD$ energies than the simpler TV_{ℓ_2}-$L1$. On the other hand, on *RubberWhale*, the opposite can be observed, with $NLTV$-$CSAD$ obtaining significantly lower error.

Method	Dim.	Hyd.	Rub.	Gro2.	Gro3.	Urb2.	Urb3.	Ven.
Multi $TV - L1$ [26]	0.1624	0.2581	0.2154	0.1561	0.7208	0.3833	0.7101	0.3947
TV_{ℓ_2}-$L1$	0.1736	0.2747	0.2062	0.1506	0.6609	0.3486	0.5028	0.3417
TV_{ℓ_2}-$CSAD$	0.1952	0.1832	0.2031	0.2115	0.8104	0.3954	1.0403	0.3486
$NLTV$-$CSAD$	0.1332	0.1934	0.1377	0.2060	0.7021	0.4098	1.1968	0.2998

Table 8: Average endpoint error (EPE) in the Middlebury dataset with public ground truth.

Some qualitative results are shown in Figure 7 for images of the MPI-Sintel database that contain large displacements, the initial seeds have been computed using DeepMatching. The first row displays, from left to right, two consecutive frames and the optical flow ground truth (color coded). The image (d) of the second row displays the ground truth occlusion map (the occluded points are shown in white). The multi-scale results obtained using the TV_{ℓ_2}-$L1$ energy are shown in (e), while the corresponding results obtained by the FALDOI strategy are shown in the third row. As it can be observed, the use of an advanced data term based on patches reduces the outlier area and better recovers the human shape. After adding the non-local regularizer the motion boundaries are more accurate and the outlier is removed.

Table 9 provides a quantitative comparison of different estimations of the optical flow obtained with different energy functionals (we average the results for 10 subsets of 10% of frames randomly selected from the final training set of the MPI-Sintel database). In order to provide for a more complete analysis of the results, the authors of MPI Sintel also proposed to evaluate different versions of the average endpoint error (EPE). When we compute the EPE for all pixels, we denote it EPE_{all} as introduced in Equation 1; if we only compute the EPE for non-occluded pixels, we have EPE_{mat}, where mat stands for matched pixels, as those which are not occluded in the current frame. Contrarily, when we only evaluate the error on occluded or unmatched areas, this is expressed as EPE_{umat} for briefness. Finally, to distinguishing how good one method performs for small, average and large displacements, the authors of MPI-Sintel introduced the measures s_{0-10}, s_{10-40} and s_{40+} which compute the EPE for pixels that have a velocity in a pre-specified range (e.g.: s_{0-10} is computed for velocities bigger or equal than 0 and strictly smaller than 10). Similarly, s_{10-40} and s_{40+}

[5]Middlebury images available from `http://vision.middlebury.edu/flow/data/`

compute the EPE for pixels with velocities in the ranges 10-40 and 40 to infinity. The velocity of a pixel is expressed in pixels per frame and can be defined as follows

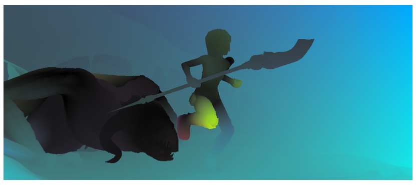
$$r(x, y) = \sqrt{u(x, y)^2 + v(x, y)^2}, \tag{7}$$

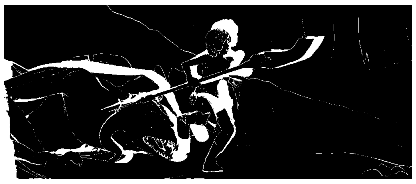
where $u(x, y)$ and $v(x, y)$ specify the horizontal and vertical flow fields of the pixel located at coordinates (x, y).

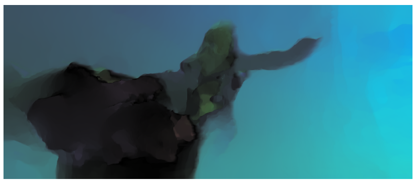
Let us notice that the energy based on the non-local Total Variation and a smooth approximation of the Census transform ($CSAD$) is the one which achieves the best results, both quantitative and qualitatively. We have also included the results obtained with 9×9 (Table 10) and 7×7 (Table 11) patch sizes. The main advantage of using a smaller patch size is the linear reduction in execution time despite obtaining a slightly worse estimation of the flow and hence a larger error. A more in depth analysis of the trade-off between speed and performance can be seen in Tables 1-6.

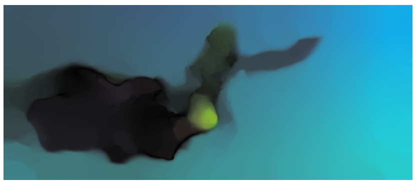
(a) First frame

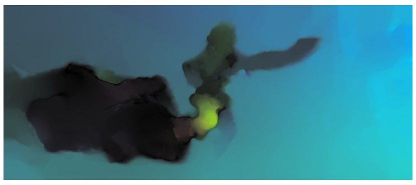
(b) Second frame


(c) Ground truth


(d) Occlusion map


(e) Multi. TV_{ℓ_2}-$L1$[26], $EPE = 3.8753$


(f) FALDOI TV_{ℓ_2}-$CSAD, EPE = 2.2078$


(g) FALDOI TV_{ℓ_2}-$L1, EPE = 2.3576$

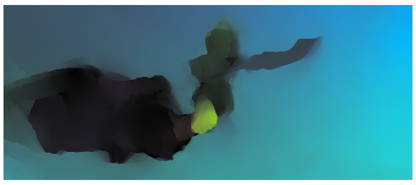
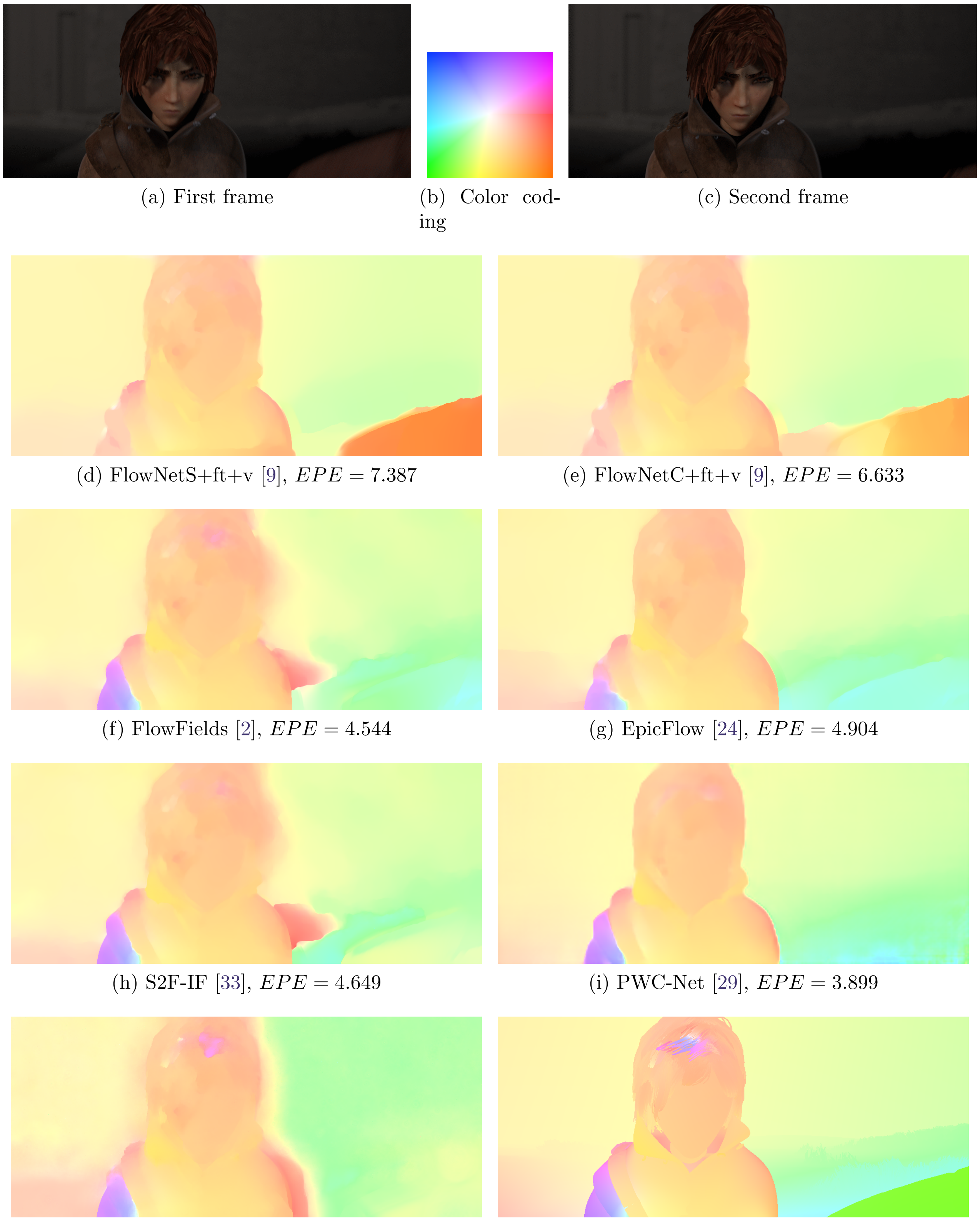

(h) FALDOI $NLTV$-$CSAD, EPE = 2.0668$

Figure 7: Comparison of FALDOI against the multiscale approach using different functionals. The initial seeds have been computed using DeepMatching.

Table 12 and Figure 8 show a quantitative and qualitative comparison, respectively, among FALDOI and several state-of-the-art methods, including methods based on the combination of sparse matches and variational techniques [5, 31] via an extra coupling term and that can also be adapted to any energy. FALDOI's results are computed with DeepMatching seeds (original image subsampled by 2) and the $NLTV$-$CSAD$ functional. The method obtains similar or better results than both FlowNet1 models in both Clean and Final passes. Compared to Epicflow [31], FALDOI obtains competing results, about a fifteen percent worse. We also compare FALDOI against the current and its co-temporary state-of-the-art. FlowFields, obtains better results in both passes and can compete with current state-of-the-ar methods like S2F-IF and PWC-Net, both methods released in late 2017.

The images in Figure 8 have been taken from the MPI-Sintel webpage. Compared to S2F-IF [33] and FlowFields [2], FALDOI better recovers the silhouettes of the girl (head, shoulder and upper arm), although it produces some halos around the girl's head and the right side of her shirt.



(a) First frame (b) Color coding (c) Second frame

(d) FlowNetS+ft+v [9], $EPE = 7.387$ (e) FlowNetC+ft+v [9], $EPE = 6.633$

(f) FlowFields [2], $EPE = 4.544$ (g) EpicFlow [24], $EPE = 4.904$

(h) S2F-IF [33], $EPE = 4.649$ (i) PWC-Net [29], $EPE = 3.899$

(j) FALDOI, $EPE = 6.928$ (k) Ground truth

Figure 8: Qualitative and quantitative comparison of different optical flow methods for a frame in the MPI-Sintel database (final pass). FALDOI uses the $NLTV$-$CSAD$ energy with DeepMatching seeds.

Functional, min.	EPEall	EPEmat.	EPEumat.	s0-10	s10-40	s40+
TV_{ℓ_2}-$L1$, multi.	6.6453	5.1181	15.8828	2.9261	8.0172	45.3955
$NLTV$-$CSAD$, multi.	8.4243	7.0860	16.8037	2.8129	9.3389	50.4899
TV_{ℓ_2}-$L1$, FALDOI min. w. part (SIFT)	7.2319	5.7715	15.8981	3.6300	8.8661	45.4429
TV_{ℓ_2}-$L1$, FALDOI min. no part. (SIFT)	7.2329	5.7530	15.7296	3.4905	8.8184	46.9513
TV_{ℓ_2}-$CSAD$, FALDOI min. (SIFT)	6.9430	5.7155	14.2069	3.5303	8.9697	46.6987
$NLTV$-$CSAD$, FALDOI min. (SIFT)	7.3158	6.1158	14.3742	3.8012	9.5212	47.5871
TV_{ℓ_2}-$L1$, FALDOI min. w. part. (Deep)	5.6260	4.1473	14.6818	2.9624	7.7698	33.2298
TV_{ℓ_2}-$L1$, FALDOI min. no part. (Deep)	5.6487	4.1213	14.8457	3.0859	7.8953	32.7651
TV_{ℓ_2}-$CSAD$, FALDOI min. (Deep)	4.6194	3.1548	13.2884	2.4288	7.3606	31.1472
$NLTV$-$CSAD$, FALDOI min. (Deep)	4.8263	3.4077	13.1599	2.6155	7.5386	32.4096

Table 9: Results for several methods based on an average of 10 randomly selected subsets of the MPI-Sintel *Final* training set. Both results with (*w. part*) and without partitions (*no part*) are shown separately.

Functional, min.	EPEall	EPEmat.	EPEumat.	s0-10	s10-40	s40+
TV_{ℓ_2}-$L1$, FALDOI min. w. part (SIFT)	7.2341	5.7638	15.9482	3.5542	8.6800	45.4984
TV_{ℓ_2}-$L1$, FALDOI min. no part. (SIFT)	7.2389	5.7520	19.5304	3.5085	8.7043	46.8969
TV_{ℓ_2}-$CSAD$, FALDOI min. (SIFT)	7.0404	5.8109	14.3061	3.4952	8.9544	46.8746
$NLTV$-$CSAD$, FALDOI min. (SIFT)	7.3300	6.1278	14.3654	3.8382	9.3472	47.6396
TV_{ℓ_2}-$L1$, FALDOI min. w. part. (Deep)	5.6818	4.1811	14.8290	2.9657	7.8635	33.0877
TV_{ℓ_2}-$L1$, FALDOI min. no part. (Deep)	5.6616	4.1535	14.9053	3.0657	7.9137	32.6208
TV_{ℓ_2}-$CSAD$, FALDOI min. (Deep)	4.6514	3.1736	13.3693	2.5764	7.2818	32.2529
$NLTV$-$CSAD$, FALDOI min. (Deep)	4.7747	3.3822	13.0989	2.7163	7.5626	31.9610

Table 10: Results similar to Table 9 but for a smaller patch size, 9×9 instead of 11×11.

Functional, min.	EPEall	EPEmat.	EPEumat.	s0-10	s10-40	s40+
TV_{ℓ_2}-$L1$, FALDOI min. w. part. (SIFT)	7.2525	5.7738	16.0008	3.5576	8.7003	45.0942
TV_{ℓ_2}-$L1$, FALDOI min. no part. (SIFT)	7.3161	5.7987	16.2589	3.6013	8.6588	46.5200
TV_{ℓ_2}-$CSAD$, FALDOI min. (SIFT)	6.9702	5.7380	14.3010	3.4938	8.7972	46.7206
$NLTV$-$CSAD$, FALDOI min. (SIFT)	7.2541	6.0569	14.3081	3.6790	9.3258	47.4345
TV_{ℓ_2}-$L1$, FALDOI min. w. part. (Deep)	5.6725	4.1920	14.7663	2.9484	7.6773	32.9252
TV_{ℓ_2}-$L1$, FALDOI min. no part. (Deep)	5.8099	4.1907	14.9475	3.0335	7.8695	32.5713
TV_{ℓ_2}-$CSAD$, FALDOI min. (Deep)	4.6294	3.1621	13.3178	2.5894	7.2129	30.9025
$NLTV$-$CSAD$, FALDOI min. (Deep)	4.7831	3.3417	13.2363	2.6508	7.5149	32.1324

Table 11: Results similar to Table 9 but for a smaller patch size, 7×7 instead of 11×11.

FlowNetS and FlowNetC [9] obtain better contours than FALDOI but they fail to properly estimate the motion of the left arm, unlike the rest of the methods. EpicFlow shows very good results, with good boundaries and not noticeable blurred regions. Finally, PWC-Net [29], the best performing method on Sintel's final pass at the time of writing, computes a very good estimation of the flow, only missing the extremely difficult motion of the girl's hair and the occluded region (bottom-left corner of the image). It is important to note that a better-performing version of FlowNet is available (FlowNet2) which significantly improves the performance of its former iteration (the one tested in this paper).

In the approach of Brox et al. [5] or Weinzaepfel et al. [31] the matches are precomputed and then added as a constraint to the energy term. Thanks to these matches the motion of small objects that disappear at the coarser scales is recovered. However, these approaches need a minimum density of sparse matches over the area of the small object in order to correctly capture large displacements,

	EPEall	EPEmat.	EPEumat.	s0-10	s10-40	s40+
Final						
$PWC-Net^{(1)}$ [29]	4.596	2.254	23.696	0.945	2.978	26.620
$S2F-IF^{(6)}$ [33]	5.417	2.549	28.795	1.157	3.468	31.262
$FlowFields^{(15)}$ [2]	5.810	2.621	31.799	1.157	3.739	33.890
$EpicFlow^{(28)}$ [24]	6.285	3.060	32.564	1.135	3.727	38.021
$FALDOI^{(41)}NLTV\text{-}CSAD$	7.337	3.580	37.904	1.487	4.355	43.526
$FALDOI^{(44)}TV_{\ell_2}\text{-}CSAD$	7.435	3.684	37.963	1.646	4.729	42.663
$FlowNetS+ft+v^{(73)}$ [9]	7.218	3.752	32.445	1.358	4.609	42.571
$FlowNetC+ft+v^{(72)}$ [9]	7.883	4.132	38.426	1.369	5.049	47.005
Clean						
$PWC-Net^{(26)}$ [29]	3.454	1.413	20.116	0.751	2.230	19.846
$S2F-IF^{(11)}$ [33]	3.500	0.988	23.986	0.524	1.976	21.960
$FlowFields^{(22)}$ [2]	3.748	1.056	25.700	0.546	2.110	23.602
$EpicFlow^{(33)}$ [24]	4.115	1.360	26.595	0.712	2.117	25.859
$FALDOI^{(43)}NLTV\text{-}CSAD$	4.927	1.542	32.535	1.047	2.647	29.719
$FALDOI^{(45)}TV_{\ell_2}\text{-}CSAD$	5.026	1.648	32.569	1.160	2.820	29.566
$FlowNetS+ft+v^{(37)}$ [9]	6.158	2.800	33.491	0.766	2.938	40.686
$FlowNetC+ft+v^{(54)}$ [9]	6.081	2.576	34.620	0.764	2.686	40.676

Table 12: MPI-Sintel test set results (27, Oct. 2018). The number inside the parentheses indicates the rank of the submission.

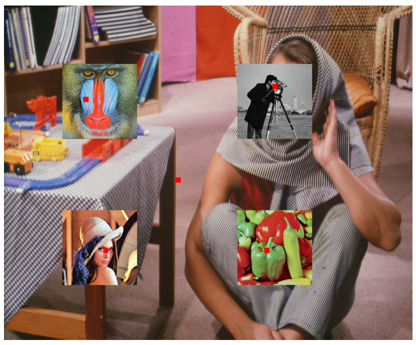
even if the matches are weighted strong enough and enough iterations are performed. By contrast, FALDOI only needs one single seed per area motion to recover the whole motion field. This is illustrated in Figure 9 where FALDOI is able to recover the four large displacements with just a single seed in each region, while none of the other methods succeed. EpicFlow[6], a very competing sparse-to-dense technique, obtains the worst results, specially analytically, showing its lack of robustness with a very limited number of seeds. Visually it is more or less capable of discerning the direction of the moving objects despite totally missing their boundaries. The multi-scale approach with $TV\text{-}L1$ energy fails to correctly estimate the flow direction and magnitude despite being able to identify the moving areas. We also include the result obtained with LDOF[7], which obtains comparable results to the latter method. In this case it has been obtained with their own seeds (using their original binary code).

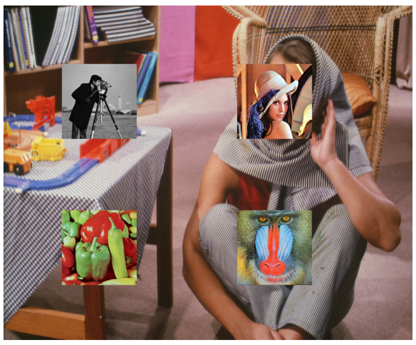
Recent optical flow datasets as MPI-Sintel [6] contain different and challenging effects, such as illumination changes, large displacements, blur, etc. In MPI-Sintel these effects have been artificially created to provide naturalistic video sequences. However, the evaluation on these datasets does not take into account the robustness to the shot noise that appears in any real sequence, being noise one of the main limitations to any imaging system. Thus, we evaluate the robustness of several approaches to noise. To this goal, we corrupt the clean images from [6] with additive white Gaussian noise for different standard deviations σ.

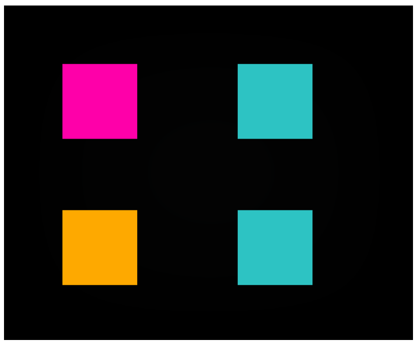
Sparse-to-dense techniques are very dependent of the initial seeds that are used to obtain a dense optical flow. It is important to note that FALDOI works even with an extremely sparse set of initial seeds. This fact allows us to choose the best matching method according to the image peculiarities, without caring that much about the density of the correspondences. In particular, for highly noisy images, SIFT correspondences are more robust than DeepMatching. Table 13 shows a comparative of FALDOI different optical flow estimation methods for different levels of Gaussian noise. We use

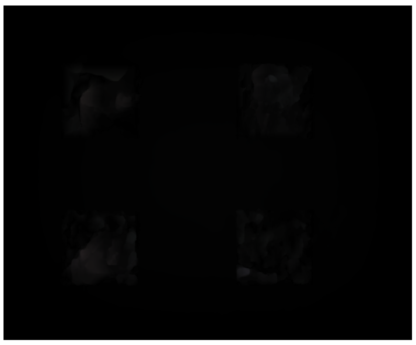
[6]EpicFlow from `https://thoth.inrialpes.fr/src/epicflow/`

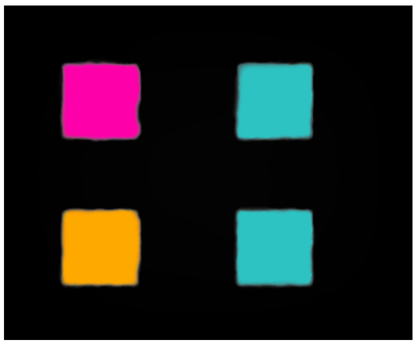
[7]Large Displacement Optical Flow from `https://www.cs.cmu.edu/~katef/LDOF.html`

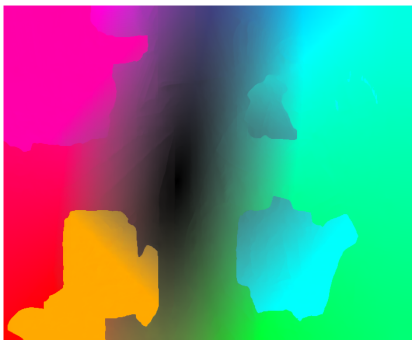

(a) First frame


(b) Second frame


(c) Ground truth


(d) Multi-scale w. TV-L1[26], $EPE = 56.2357$


(e) FALDOI [23] TV-L1 energy, $EPE = 15.5351$


(f) EpicFlow [24], $EPE = 348.9911$

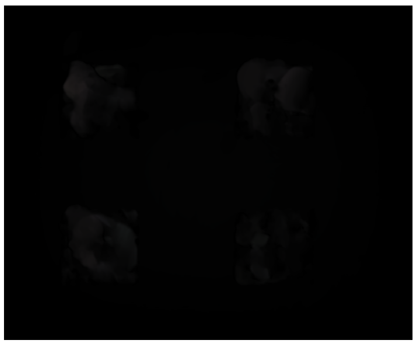

(g) LDOF [5], own seeds, $EPE = 56.7676$

Figure 9: Large displacement on a composition made of the images *Barbara*, *Baboon*, *Cameraman*, *Lena* and *Peppers*. It contains large diagonal translations and a slight deformation of the background. Five initial seeds have been manually selected and are shown in (a).

the TV_{ℓ_2}-$L1$ functional with SIFT matches. As shown in the table, the multi-scale method provides the best optical flow estimation in noisy images. For noise levels of standard deviation greater or equal to 10 and 20 respectively, FALDOI produces better results than EpicFlow and DeepFlow.

5 Conclusions and Future Work

We have presented an in-depth analysis of the FALDOI method for optical flow estimation. The method works at the original image scale and finds a good local minimum of any optical flow energy functional using an adaptive coordinate descent strategy guided by a sparse set of initial matches. This is a general technique that consistently outperforms the multi-scale strategy for the same energy

σ	10	20	30	40
DeepFlow without matches (pure multi-scale)	0.7468	1.0353	1.3832	1.4808
DeepFlow [31]	0.7766	1.5796	2.6128	4.3918
EpicFlow [24]	1.1418	1.6654	2.3386	3.2328
FALDOI (*TV*-L_1) with SIFT matches	0.9486	1.5634	2.1090	2.9830

Table 13: Results over a set of five samples from the MPI-Sintel clean dataset with different standard deviation (σ) levels of Gaussian noise.

functional. With respect to alternative techniques that also include sparse matches in an energy functional, the performance is comparable to DeepFlow [31] and superior to LDOF [5] while being more robust to a low density of matches, high levels of noise and outliers in the matches. The only requirement is that at least one correct match is given for each object in motion. For best overall results, we propose to use an energy with advanced data and regularization terms, namely, a smooth variant of the Census transform with a non-local TV regularization, providing robustness to illumination changes and occlusions while handling motion discontinuities.

We present accurate quantitative and qualitative results that are comparable with state-of-the-art methods. We have also compared the method against the current state of the art techniques, which mainly use CNNs to learn to estimate the optical flow, and are specially fit to handle more complex and realistic scenarios (e.g., the final pass in MPI-Sintel) where edge-refining post-processing techniques are not viable since the edges are corrupted by noise and motion blur. The main contribution presented in this paper is the speedup optimization through the use of OpenMP parallel sentences and also the usage of image partitions to process different parts of the image at the same time. The final reduction in execution time is pretty significant and makes the method more useful for applications where run-time is fairly critical.

As future work we plan to extend the joint occlusion estimation to the other functionals (currently, only TV_{ℓ_2}-$L1$ is included) to reduce the halo effect in occluded regions. In addition, the best-performing functional, $NLTV$-$CSAD$, could be fairly improved in terms of speed, given its nature, by developing a scheme that exploits GPU parallelization (the same applies to TV_{ℓ_2}-$CSAD$).

A Minimizing the Energy

The numerical minimization algorithm for the general energy (2) is obtained in this paper by decoupling both terms. We linearize the image I_{t+1} near a given optical flow $\mathbf{u}_0 = (u_{0,1}, u_{0,2})$ and make the following approximation $I_{t+1}(\mathbf{x} + \mathbf{u}(x)) \approx I^{lin}_{t+1}(\mathbf{x} + \mathbf{u}(\mathbf{x}))$, where

$$\begin{aligned} I^{lin}_{t+1}(\mathbf{x} + \mathbf{u}(\mathbf{x})) =& I_{t+1}(\mathbf{x} + \mathbf{u}_0(\mathbf{x})) \\ &+ I^{x}_{t+1}(\mathbf{x} + \mathbf{u}_0(\mathbf{x}))(u_1 - u_{0,1})(\mathbf{x}) \\ &+ I^{x}_{t+1}(\mathbf{x} + \mathbf{u}_0(\mathbf{x}))(u_2 - u_{0,2})(\mathbf{x}), \end{aligned}$$

and I^x_{t+1}, I^y_{t+1} denote the partial derivatives of I_{t+1} with respect to $\mathbf{x}$ and $\mathbf{y}$ respectively. Let us recall that the two data terms E_D that we have considered in Section 4.1 depend on $I_t(\mathbf{x})$ and $I_{t+1}(\mathbf{x}+\mathbf{u}(\mathbf{x}))$; we will denote as $E_{D,lin}$ the same data term but depending on $I_t(\mathbf{x})$ and $I^{lin}_{t+1}(\mathbf{x} + \mathbf{u}(\mathbf{x}))$. In order to decouple the fidelity term $E_{D,lin}(u)$ and the regularization term $E_R(\mathbf{u})$ in (2), we introduce an auxiliary variable $\mathbf{v}$ representing the optical flow and we penalize its deviation from $\mathbf{u}$. Thus, the energy to minimize is

$$J(\mathbf{u}, \mathbf{v}) = J_{D,lin}(\mathbf{v}) + \beta J_R(\mathbf{u}) + \frac{1}{2\theta} \int_\Omega \|\mathbf{u} - \mathbf{v}\|^2, \tag{8}$$

depending on the two variables $\mathbf{u}, \mathbf{v}$, where $\theta > 0$. The decoupled energy (8) can be minimized by an alternating minimization procedure; alternatively fixing one variable and minimizing with respect to the other one. Section 4.1 presents the different possibilities for the energy.

1. For $\mathbf{v}$ fixed, let us consider each of the two different regularization terms, $J_R^1(\mathbf{u})$ and $J_R^2(\mathbf{u})$, presented in Section 4.1.

   1.1. In the case of $J_R^1(\mathbf{u})$, we reformulate the problem as a min-max problem incorporating the dual variables. Then, the minimization problem can be solved as a saddle-point problem. Following the notation of Osher et al [11], for $\mathbf{v} = (v_1, v_2)$ fixed, we solve

   $$\int_\Omega \int_\Omega \omega(\mathbf{x}, \mathbf{y})(u_i(\mathbf{x}) - u_i(\mathbf{y}))p(\mathbf{x}, \mathbf{y}) d\mathbf{y} d\mathbf{x} + \frac{1}{2\theta} \int_\Omega (u_i - v_i)^2 \, d\mathbf{x}, \tag{9}$$

   for $i = 1, 2$, and p is the dual variable defined on $\Omega \times \Omega$. Let us explain it in detail. First, it is necessary to extend the notion of derivatives to a non-local framework. The non-local derivative can be written as

   $$\partial_y u_i(\mathbf{x}) = \frac{u_i(\mathbf{x}) - u_i(\mathbf{y})}{d(\mathbf{x}, \mathbf{y})}, \tag{10}$$

   where $d(\mathbf{x}, \mathbf{y})$ is a positive measure between two points $\mathbf{x}, \mathbf{y}$. By taking $d(\mathbf{x}, \mathbf{y})$ such that $w(\mathbf{x}, \mathbf{y}) = d(\mathbf{x}, \mathbf{y})^{-2}$, the non-local gradient $\nabla_w u_i(\mathbf{x}, \mathbf{y})$ is defined as the vector of all partial derivatives

   $$\nabla_w u_i(\mathbf{x}, \mathbf{y}) = (u_i(\mathbf{x}) - u_i(\mathbf{y}))\sqrt{w(\mathbf{x}, \mathbf{y})} \quad \mathbf{x}, \mathbf{y} \in \Omega. \tag{11}$$

   Now, by writing $\vec{p} := p(\mathbf{x}, \mathbf{y})$ for $(\mathbf{x}, \mathbf{y}) \in \Omega \times \Omega$, the non-local divergence $div_w \vec{p}(\mathbf{x})$ is defined as the adjoint of the non-local gradient

   $$\mathrm{div}_w \vec{p}(\mathbf{x}) = \int_\Omega (p(\mathbf{x}, \mathbf{y}) - p(\mathbf{y}, \mathbf{x}))\sqrt{w(\mathbf{x}, \mathbf{y})} d\mathbf{y}. \tag{12}$$

   Definition 1. *The solution of (9) is given by the following iterative scheme*

   $$\begin{aligned} p(\mathbf{x}, \mathbf{y})^{n+1} &= \frac{p(\mathbf{x}, \mathbf{y})^n + \tau(\overline{u}_i^n(\mathbf{x}) - \overline{u}_i^n(\mathbf{y})\sqrt{w(\mathbf{x}, \mathbf{y})})}{1 + \tau|\nabla_w u_i(\mathbf{x}, \mathbf{y})|}, & (13)\\ u_i^{n+1}(\mathbf{x}) &= u_i^n(\mathbf{x}) - \sigma\left(\frac{(u_i^n(\mathbf{x}) - v_i(\mathbf{x}))}{\theta} - \mathrm{div}_w \vec{p}(\mathbf{x})\right), & (14)\\ \overline{u}_i^{n+1}(\mathbf{x}) &= 2u_i^{n+1}(\mathbf{x}) - u_i^n(\mathbf{x}), & (15) \end{aligned}$$

   where u_i is the primal variable and $\vec{p}$ is the dual variable.

   1.2. In the case of $J_R^2(\mathbf{u})$ we use the primal-dual algorithm that Chambolle proposed to minimize the ROF model [7] and which is based on a dual formulation of the TV. Then, the minimization problem can be solved as a saddle-point problem. For $\mathbf{v}$ fixed, we solve

   $$\min_{\mathbf{u}} \max_{\xi} \int_\Omega \langle D\mathbf{u}, \xi \rangle d\mathbf{x} + \int_\Omega \frac{1}{2\theta} \|\mathbf{u} - \mathbf{v}\|)^2 d\mathbf{x}, \tag{16}$$

   where the dual variables are $\xi = \begin{pmatrix} \xi_{11} & \xi_{12} \\ \xi_{21} & \xi_{22} \end{pmatrix}$ and satisfy $||\xi||_F \leq 1$.

Definition 2. *The solution of (16) is given by the following iterative scheme*

$$\xi_{i1}^{n+1} = \frac{\xi_{i1}^n + \tau \overline{u}_{ix}^n}{\max(1, ||\xi||_2)}, \qquad \xi_{i2}^{n+1} = \frac{\xi_{i2}^n + \tau \overline{u}_{iy}^n}{\max(1, ||\xi||_2)}, \tag{17}$$

$$u_i^{n+1} = u_i^n - \sigma \left(\frac{(u_i^n - v_i)}{\theta} - \operatorname{div}(\xi_{i1}^n, \xi_{i2}^n) \right), \tag{18}$$

$$\overline{u}_i^{n+1} = 2u_i^{n+1} - u_i^n, \tag{19}$$

where $i = 1, 2$.

2. For $\mathbf{u}$ fixed, let us consider each of the two different data terms, $J_D^1(\mathbf{v})$ and $J_D^2(\mathbf{v})$, presented in Section 4.1.

   2.1. Case $J_D^1(\mathbf{v})$, Li and Osher [18] present a simple algorithm to find the optimal value of the function $E(x) = \sum_i^n w_i |x - a_i| + F(x)$ when the w_i are non-negative and F is strictly convex. If F is also differentiable and F' is bijective, it is possible to obtain an explicit formula in terms of the median. For $\mathbf{u}$ fixed, we solve

$$\int_\Omega C(\mathbf{v}, \mathbf{x}) d\mathbf{x} + \frac{1}{2\theta} \int_\Omega \|\mathbf{u} - \mathbf{v}\|^2 d\mathbf{x}. \tag{20}$$

   Following the ideas of [30], we solve the discrete version of this problem. Due to the isotropy of the quadratic term, the optimal solution of $C(\mathbf{v}, \mathbf{x})$ can be obtained solving a one dimensional problem. In particular, setting $\mathbf{v} = \hat{\mathbf{v}} + \delta \frac{\nabla I(x+v_o)}{|\nabla I(x+v_o)|} + \delta \frac{\nabla^+ I(x+v_o)}{|\nabla^+ I(x+v_o)|}$ being $\nabla^+ I$ an orthogonal vector to the gradient, where δ is the new variable. Then, we minimize over δ

$$\frac{1}{2\tau} \delta^2 + \lambda \int_\Omega |\nabla I_{t+1}(\mathbf{x} + \hat{\mathbf{v}}_o)| \, |G(\hat{\mathbf{v}}) + \delta| \, d\mathbf{y}, \tag{21}$$

   where

$$G_{\mathbf{y}}(\hat{\mathbf{v}}) = \frac{I_t(\mathbf{x}) - I_t(\mathbf{y}) - I_{t+1}(\mathbf{x} + \hat{\mathbf{v}}_o) + I_{t+1}(\mathbf{y} + \hat{\mathbf{v}}_o)}{|\nabla I_{t+1}(\mathbf{x} + \hat{\mathbf{v}}_o)|} + \frac{(\hat{\mathbf{v}} - \hat{\mathbf{v}}_o)^T \nabla I_{t+1}(\mathbf{x} + \hat{\mathbf{v}}_o)}{|\nabla I_{t+1}(\mathbf{x} + \hat{\mathbf{v}}_o)|}.$$

   Definition 3. *The minimum of (21) with respect to δ is*

$$\delta^* = \operatorname{median}\{b_1, \ldots, b_n, a_0, \ldots a_n\} \tag{22}$$

   where $b_i = -G_i(\hat{\mathbf{v}})$ and $a_i = (n - 2i)\lambda |\nabla I_{t+1}(\mathbf{x} + \mathbf{v}_o)|$ for all the discrete neighbors i (corresponding to $\mathbf{y}$ above), where n is the number of points in the discrete neighborhood.

   2.2. Case $J_D^2(\mathbf{v})$. Notice that this term is a particular case of the previous data term. The functional to minimize

$$\int_W \lambda |\rho(\mathbf{v})| + \frac{1}{2\theta} \int_W \|\mathbf{u} - \mathbf{v}\|^2 d\mathbf{x}, \tag{23}$$

   where $\rho(\mathbf{v}) = I_{t+1}(\mathbf{x}, \mathbf{v}_o) + \langle \nabla I_{t+1}(\mathbf{x} + \mathbf{v}_o), (\mathbf{v} - \mathbf{v}_o) \rangle - I_t(\mathbf{x})$, does not depend on spatial derivatives on $\mathbf{v}$. Then, a simple thresholding step gives an explicit solution [35].

   Definition 4. *The minimum of (23) with respect to $\mathbf{v}$ is*

$$\mathbf{v} = \mathbf{u} + \begin{cases} \lambda\theta \nabla I_{t+1} & \text{if } \rho(\mathbf{u}) < -\lambda\theta |\nabla I_{t+1}|^2, \\ -\lambda\theta \nabla I_{t+1} & \text{if } \rho(\mathbf{u}) > \lambda\theta |\nabla I_{t+1}|^2, \\ -\rho(\mathbf{u}) \frac{\nabla I_{t+1}}{|\nabla I_{t+1}|^2} & \text{if } |\rho(\mathbf{u})| \le \lambda\theta |\nabla I_{t+1}|^2. \end{cases} \tag{24}$$

B Implementation Details

Our code is based on the original code from [23] and is written in C/C++. Image warpings use bicubic interpolation. The image gradient is computed using centered-derivatives. Input images have been normalized between $[0, 1]$. The algorithm parameters are initialized with the same default setting for all the experiments. Both time steps are set to $\tau = \sigma = 0.125$ to ensure convergence. As stopping criterion, the optical flow algorithm uses the infinite-norm of the difference between two consecutive values of u with a threshold of 0.01. The coupling parameter θ is set to 0.3. The smoothness term weight β is set to 1/40 for the TV_{ℓ_2}-$L1$ functional and to $\beta = \frac{N-1}{80}$ for the $NLTV$-$CSAD$ one, as suggested by [30], where N is the cardinality of the neighborhood considered in the $CSAD$ term (we use a neighborhood of 7×7 pixel size in the data term and then $N = 49$) and we fixed $\sigma_c = 2$ and $\sigma_s = 2$ for the spatial an color domain of the $NLTV$ term. For the `iterated FALDOI` strategy we set MAX_IT to 3. The size of the patch Ω^{patch} in the local minimization is 11×11 pixels. When partitions are used ($split\_img$ parameter set to 1), by default we set $h_{parts} = 3$ and $v_{parts} = 2$ (which implies 6 partitions per image). At the time of writing, only the TV_{ℓ_2}-$L1$ energy is compatible with the partitioning scheme presented before. We manage all the variables involved in the minimization through a vector of structs with length equal to the number of partitions used.
It is important to notice that we have generated all the results of this paper with a forward-backward consistency check threshold $\epsilon = 2$. Nevertheless, further testing during the submission of this article comparing the End Point Error and the selected threshold, suggested that other values gave lower error rates. These tests were analyzed with the entire MPI-Sintel Dataset (final and clean pass) and average and median results were plotted. Moreover, the optimal values were different for each matcher: SIFT benefited of smaller (more restrictive) thresholds, minimum error with $\epsilon = 0.45$; while DeepMatching yielded better results with bigger values, minimum error with $\epsilon = 12.55$. As a consequence, in the online demo, the default values are the ones which yield the minimum error for the approximated range specified above.
In the online demo, four parameters can be tuned: 1) the matcher algorithm used (SIFT or DeepMatching); 2) the patch radius ($patch\_size = 2 * patch\_radius + 1$), assuming square patches. The allowed values are those tested throughout this article, that is: 3, 4 or 5 (equivalent to 7×7, 9×9 and 11×11 patch sizes, respectively). The default value used in this paper is 5. However, in the online demo we set 4 as default instead to ensure that the code can executed in less than a minute; 3) the number of local iterations (local minimization), each one following the steps in Algorithm 6. In Algorithm 5 (FALDOI algorithm end-to-end with pruning), the number of local iterations is referred to as MAX-IT. By default, 3 iterations are performed but one may choose only 2 for a faster (but worse) estimation; 4) the number of iterations for each patch visited in each local iteration. Define how many times step 5 of Algorithm 6 ($\mathbf{u} \leftarrow \texttt{flow-refinement}(A, B, \Omega_{\mathbf{y}}, E, \mathbf{u})$) is repeated to estimate the energy over the patch Ω_y. In the demo, the default and recommended value is 4 (used for all results in this paper) but one may also use 3 to reduce the run-time despite losing estimation accuracy. It is also important to notice that the only available energy functional in the online demo is the TV_{ℓ_2}-$L1$ due to the execution time limits of IPOL. Moreover, the partition scheme described in Section 3 is always used in the demo so as to reduce the execution time further.

Acknowledgment

The authors acknowledge support by TIN2015-70410-C2-1-R(MINECO/FEDER, UE) project and by H2020-MSCA-RISE-2017 project, reference 777826 NoMADS.

Image Credits

All images with the exception of the one in Figure 9 (which has been generated by us) have been extracted from the MPI-Sintel Dataset at http://sintel.is.tue.mpg.de/downloads.

References

[1] A. Ayvaci, M. Raptis, and S. Soatto, *Sparse occlusion detection with optical flow*, International Journal of Computer Vision, 97 (2012), pp. 322–338. https://doi.org/10.1007/s11263-011-0490-7.

[2] C. Bailer, B. Taetz, and D. Stricker, *Flow fields: Dense correspondence fields for highly accurate large displacement optical flow estimation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, (2018). https://doi.org/10.1109/TPAMI.2018.2859970.

[3] S. Baker, D. Scharstein, J.P. Lewis, S. Roth, M.J. Black, and R. Szeliski, *A database and evaluation methodology for optical flow*, International Journal of Computer Vision, 92 (2011), pp. 1–31. https://doi.org/10.1007/s11263-010-0390-2.

[4] C. Ballester, L. Garrido, V. Lazcano, and V. Caselles, *A TV-L1 Optical Flow Method with Occlusion Detection*, vol. 7476 of Lectures Notes in Computer Science, 2012, p. 3140. https://doi.org/10.1007/978-3-642-32717-9_4.

[5] T. Brox and J. Malik, *Large displacement optical flow: descriptor matching in variational motion estimation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 33 (2011), pp. 500–513. https://doi.org/10.1109/TPAMI.2010.143.

[6] D.J. Butler, J. Wulff, G.B. Stanley, and M.J. Black, *A naturalistic open source movie for optical flow evaluation*, in Proceedings of European Conference on Computer Vision (ECCV), 2012, pp. 611–625. https://doi.org/10.1007/978-3-642-33783-3_44.

[7] A. Chambolle and T. Pock, *A first-order primal-dual algorithm for convex problems with applications to imaging*, Journal of Mathematical Imaging and Vision, 40 (2011), pp. 120–145. https://doi.org/10.1007/s10851-010-0251-1.

[8] P. Dollár and C.L. Zitnick, *Structured forests for fast edge detection*, in Proceedings of IEEE International Conference on Computer Vision (ICCV), 2013, pp. 1841–1848. https://doi.org/10.1109/ICCV.2013.231.

[9] A. Dosovitskiy, P. Fischery, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Smagt, D. Cremers, and T. Brox, *Flownet: Learning optical flow with convolutional networks*, in Proceedings of IEEE International Conference on Computer Vision (ICCV), IEEE Computer Society, pp. 2758–2766. http://dx.doi.org/10.1109/ICCV.2015.316.

[10] J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani, *Pathwise coordinate optimization*, tech. report, Annals of Applied Statistics, 2007.

[11] G. Gilboa and S. Osher, *Nonlocal operators with applications to image processing*, Multiscale Modeling & Simulation, 7 (2008), pp. 1005–1028. https://doi.org/10.1137/070698592.

[12] D. Hafner, O. Demetz, and J. Weickert, *Why is the census transform good for robust optic flow computation?*, in Scale-Space and Variational Methods in Computer Vision, 2013, pp. 210–221. https://doi.org/10.1007/978-3-642-38267-3_18.

[13] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, *Flownet 2.0: Evolution of optical flow estimation with deep networks*, in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jul 2017. https://doi.org/10.1109/CVPR.2017.179.

[14] J. Kannala and S.S. Brandt, *Quasi-dense wide baseline matching using match propagation.*, in Proc. IEEE Conference on Computer Vision and Pattern Recognition, 2007, pp. 1–8.

[15] M. Leordeanu, R. Sukthankar, and C. Sminchisescu, *Efficient closed-form solution to generalized boundary detection*, in Proceedings of European Conference on Computer Vision (ECCV), 2012, pp. 516–529. https://doi.org/10.1007/978-3-642-33765-9_37.

[16] M. Leordeanu, A. Zanfir, and C. Sminchisescu, *Locally affine sparse-to-dense matching for motion and occlusion estimation*, in Proceedings of IEEE International Conference on Computer Vision (ICCV), 2013, pp. 1721–1728. https://doi.org/10.1109/ICCV.2013.216.

[17] M. Lhuillier and L. Quan, *Match propagation for image-based modeling and rendering*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 24 (2002), pp. 1140–1146. https://doi.org/10.1109/TPAMI.2002.1023810.

[18] Y. Li and S. Osher, *A new median formula with applications to PDE based denoising*, Communications in Mathematical Sciences, 7 (2009), pp. 741–753. https://doi.org/10.4310/CMS.2009.v7.n3.a11.

[19] D.G. Lowe, *Object recognition from local scale-invariant features*, in Proceedings of International Conference on Computer Vision (ICCV), 1999. https://doi.org/10.1109/ICCV.1999.790410.

[20] M. Menze, C. Heipke, and A. Geiger, *Discrete optimization for optical flow*, in Pattern Recognition. DAGM, vol. 9358 of Lecture Notes in Computer Science, 2015, pp. 16–28. https://doi.org/10.1007/978-3-319-24947-6_2.

[21] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool, *An Algorithm for Total Variation Minimization and Applications*, International Journal of Computer Vision, 65 (2005), pp. 43–72. https://doi.org/10.1023/B:JMIV.0000011325.36760.1e.

[22] O. Martorell Nadal, *Sparse to dense optical flow with occlusion estimation*, master's thesis, Master in Computer Vision, Computer Vision Center, UAB Campus, September 2017.

[23] R.P. Palomares, E. Meinhardt-Llopis, C. Ballester, and G. Haro, *FALDOI: A New Minimization Strategy for Large Displacement Variational Optical Flow*, Journal of Mathematical Imaging and Vision, 58 (2017), pp. 27–46. https://doi.org/10.1007/s10851-016-0688-y.

[24] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid, *EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow*, in Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition (CVPR), 2015. https://doi.org/10.1109/CVPR.2015.7298720.

[25] Ives Rey Otero and Mauricio Delbracio, *Anatomy of the SIFT Method*, Image Processing On Line, 4 (2014), pp. 370–396. https://doi.org/10.5201/ipol.2014.82.

[26] J. Sánchez Pérez, E. Meinhardt-Llopis, and G. Facciolo, *TV-L1 Optical Flow Estimation*, Image Processing On Line, 3 (2013), pp. 137–150. https://doi.org/10.5201/ipol.2013.26.

[27] F. Stein, *Efficient computation of optical flow using the census transform*, in Pattern Recognition. DGAM, 2004, pp. 79–86. https://doi.org/10.1007/978-3-540-28649-3_10.

[28] E. Strekalovskiy, A. Chambolle, and D. Cremers, *Convex relaxation of vectorial problems with coupled regularization*, SIAM Journal on Imaging Sciences, 7 (2014), pp. 294–336. https://doi.org/10.1137/130908348.

[29] D. Sun, X. Yang, M-Y. Liu, and J. Kautz, *PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume*, in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

[30] C. Vogel, S. Roth, and K. Schindler, *An evaluation of data costs for optical flow*, in Pattern recognition. GCPR, vol. 8142 of Lectures Notes in Computer Science, 2013, pp. 343–353. https://doi.org/10.1007/978-3-642-40602-7_37.

[31] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid, *DeepFlow : Large displacement optical flow with deep matching*, in Proceedings of IEEE International Conference on Computer Vision (ICCV), 2013. https://doi.org/10.1109/ICCV.2013.175.

[32] M. Werlberger, T. Pock, and H. Bischof, *Motion estimation with non-local total variation regularization*, in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2010, pp. 2464–2471. https://doi.org/10.1109/CVPR.2010.5539945.

[33] Y. Yang and S. Soatto, *S2F: Slow-to-fast Interpolator Flow*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (2017). https://doi.org/10.1109/CVPR.2017.401.

[34] R. Zabih and J. Woodfill, *Non-parametric local transforms for computing visual correspondence*, in Proceedings of European Conference on Computer Vision (ECCV), 1994, pp. 151–158. https://doi.org/10.1007/BFb0028345.

[35] C. Zach, T. Pock, and H. Bischof, *A Duality Based Approach for Realtime TV-L1 Optical Flow*, in Pattern Recognition. DAGM, vol. 4713 of Lecture Notes in Computer Science, 2007, pp. 214–223. https://doi.org/10.1007/978-3-540-74936-3_22.