J Real-Time Image Proc
DOI 10.1007/s11554-014-0415-0

SPECIAL ISSUE PAPER

Video Extruder: a semi-dense point tracker for extracting beams of trajectories in real time

Matthieu Garrigues · Antoine Manzanera · Thierry M. Bernard

Received: 25 July 2013 / Accepted: 12 March 2014

Abstract Two crucial aspects of general-purpose embedded visual point tracking are addressed in this paper. First, the algorithm should reliably track as many points as possible. Second, the computation should achieve real-time video processing, which is challenging on low power embedded platforms. We propose a new multi-scale semi-dense point tracker called *Video Extruder*, whose purpose is to fill the gap between short-term, dense motion estimation (optical flow) and long-term, sparse salient point tracking. This paper presents a new detector, including a new salience function with low computational complexity and a new selection strategy that allows to obtain a large number of keypoints. Its density and reliability in mobile video scenarios are compared with those of the FAST detector. Then, a multi-scale matching strategy is presented, based on hybrid regional coarse-to-fine and temporal prediction, which provides robustness to large camera and object accelerations. Filtering and merging strategies are then used to eliminate most of the wrong or useless trajectories. Thanks to its high degree of parallelism, the proposed algorithm extracts beams of trajectories from the video very efficiently. We compare it with the state-of-the-art pyramidal Lucas–Kanade point tracker and show that, in short range mobile video scenarios, it yields similar quality results, while being up to one order of magnitude faster. Three different parallel implementations of this tracker are presented, on multi-core CPU, GPU and ARM SoCs. On a commodity 2010 CPU, it can track 8,500 points in a 640 × 480 video at 150 Hz.

Keywords Point tracking · Optical flow · Beam of trajectories · Semi-dense · Real time

1 Introduction

Estimating the apparent displacement of points in a video sequence of a dynamic scene is a fundamental primitive in many applications of computer vision. In real-time systems, such a primitive must naturally be much faster than the processing rate.

A notable difficulty of motion estimation is the measure of reliability: how trustworthy are the estimated velocities? On the other hand, it is often desirable to get the densest possible estimation, i.e. computing the displacement for (almost) every point in the image. This occurs when segmenting motion [21] or when spatial statistics need to be computed on velocities, e.g. to extract dominant planes by cumulative structure from motion [2] or to recognise actions [5]. In applications using long-term trajectory estimation (e.g. activity recognition), dense spatial sampling also performs better than sparse trajectories of interest points [25].

However, there is a certain antagonism between reliability and density and, practically, one has to choose between sparse tracking or dense optical flow. Tracking algorithms aim at providing reliable motion parameters for a reduced set of points, using spatial selection and long-term temporal analysis. Optical flow algorithms aim at providing an acceptable estimation of velocity for every pixel, using short-term point matching and spatial regularisation of the motion field.

M. Garrigues · A. Manzanera (✉) · T. M. Bernard
ENSTA-ParisTech, 828 Boulevard des Maréchaux,
91762 Palaiseau Cedex, France
e-mail: antoine.manzanera@ensta.fr;
antoine.manzanera@ensta-paristech.fr
URL: http://www.ensta-paristech.fr

M. Garrigues
e-mail: matthieu.garrigues@ensta-paristech.fr

T. M. Bernard
e-mail: thierry.bernard@ensta-paristech.fr

Published online: 01 April 2014

The origins of such dichotomy are well known: the aperture problem subordinates motion estimation to the existence of salient structures whereas the piecewise continuity of the projected velocity field requires spatial smoothness. But this incompatibility is only apparent, since salience or smoothness are not intrinsic to the physical object, but related to the scale of analysis. In practice, the distinction is mostly driven by implementation choices: is it better to put the computational effort in spatial selection and extraction of descriptors for long-term prediction and tracking, or to put it in spatial regularisation for globally estimating the motion field? The choice, which limits the versatility of the system, may depend on the application.

As an attempt to fill the gap between sparse tracking and dense optical flow, we propose an intermediate approach that performs long-term tracking of a set of points (particles), designed to be as dense as possible. It results in a versatile motion estimation primitive, in that it can provide both a beam of trajectories with temporal consistency and a field of displacements with spatial consistency. It is designed to be almost fully parallel, thus extremely fast on multi-core chips such as nowadays GPUs or multi-core CPUs.

There exist different real-time compatible algorithms for sparse point tracking, e.g. Tomasi and Kanade [24], Fassold et al. [9], and for dense optical flow, e.g. d'Angelo et al. [6], but these techniques do not provide the level of versatility we are looking for. Parallelization of motion estimation algorithms has been addressed using different models and architectures: Botella et al. [1] implemented a bio-inspired dense optical flow on FPGA. Sinha et al. [22, 23] ported a sparse feature tracking and matching on GPU. Rabe et al. [15] used a GPU-based optical flow to build a dense 3D motion field, whereas Doyle et al. [7] offloaded only the corner detector to the GPU.

Our approach is closer to the *Particle Video* algorithm of Sand and Teller [19] yet not based on dense optical flow algorithm. We use temporal and spatial coarse-to-fine prediction and filtering for each particle but no explicit spatial filtering. Eventually, our method is real-time by design. Following [19], we use the term "particles" to refer to trackable image points in space-time. This meaning differs from the one used in "particle filter" methods, where they designate a sample point in a state space, which may globally represent an object in tracking methods [12].

The contributions of our work are the following. We introduce a new point detector called MIEL, whose goal is to eliminate only the points whose matching will be ambiguous. It provides the semi-dense candidate particle field. We propose an evaluation method of point detectors adapted to short range fast video. We propose a massively parallel matching algorithm based on hybrid temporal and coarse-to-fine spatial prediction, which is robust to large camera and object acceleration. In our method, spatial consistency is not enforced by explicit spatial filtering but only used as part of a test to reject unreliable particle matching. Finally, we show real-time implementations on three different platforms: multi-core CPU, GPU, and embedded ARM processor.

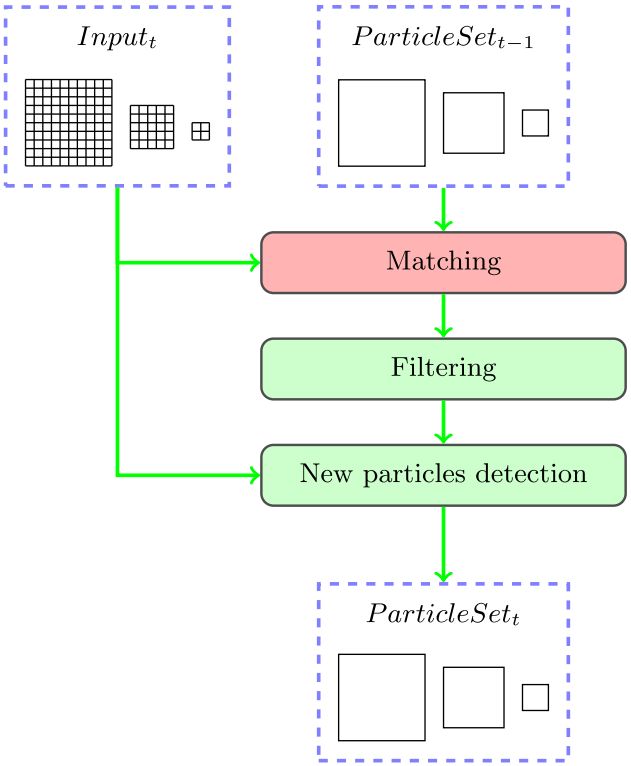

Fig. 1 Overview of the tracker. Only the matching operator (*in red*) has to be run every frame. The filtering and detection operations may be less frequent. In our applications, they are typically triggered every five frames

The proposed tracking procedure is split into three main steps: matching, filtering and new particle detection. Figure 1 shows an overview of the tracker, whose inputs are the current frame of the video stream and the (multi-resolution) particle set at time $t - 1$. The matching step function is to locate each particle in the new frame, and remove occluded particles. The filtering step uses heuristics to check the validity of each particle state and to remove invalid particles. The last step extracts new particles from the current image.

The paper is organised as follows. Section 2 presents our study on the weak keypoint selection algorithms and defines the feature vector used for matching. Section 3 details our pyramidal tracking algorithm, the filter-merge procedures, and presents a comprehensive evaluation of our algorithm, compared with the OpenCV LK tracker [3] and Farnebäck's dense optical flow algorithm [8]. Finally Sect. 4 discusses implementation issues and presents time benchmarks obtained with different architectures.

2 Weak keypoint selection and description

The goal of keypoint detection is to provide a high spatial density of particles, wherever possible, while avoiding points whose matching may be ambiguous (homogeneous zones, straight edges). To this end, we select weak keypoints. Compared to dense regular sampling, it saves a lot of useless computational effort on points whose matching is either very costly or unreliable, often both.

To get a particle field that is as dense as possible, the detector is better applied at different scales. We choose to apply the same single scale detector on a dyadic image pyramid. Section 2.1 details the classical FAST detector [17] and our own MIEL detector, which is an improvement of previous works [10]. Both are computed on the same spatial support B_3 (see Fig. 7),

To assess the different detectors with respect to our requirements, we compare (1) the number of keypoints and (2) the matching errors made by the tracking algorithm on these keypoints. We present our evaluation method and discuss the results in Sect. 2.2. The repeatability property is usually given much importance [20]. But it does not make much sense here. First because with a high number of keypoints, the repeatability should always be close to 100 %. Second, the detector is only applied to create new particles: an existing particle is matched to a position, not to another particle.

A description vector must be attached to each particle to be used for comparison purposes by the matching procedure. This descriptor is defined in Sect. 2.3.

2.1 Detectors

2.1.1 The FAST detector

The FAST [16] detector computes local statistics using $B_3(\mathbf{p})$, the Bresenham circle of radius 3 (Fig. 7, left) centred on the candidate keypoint $\mathbf{p}$. Let I be the grayscale image. The detection computes the two sets S^- and S^+:

- $S_t^-(\mathbf{p}) = \{\mathbf{q} \in \mathbf{B_3}(\mathbf{p}); \mathbf{I}(\mathbf{q}) \leq \mathbf{I}(\mathbf{p}) - \mathbf{t}\}$
- $S_t^+(\mathbf{p}) = \{\mathbf{q} \in \mathbf{B_3}(\mathbf{p}); \mathbf{I}(\mathbf{q}) \geq \mathbf{I}(\mathbf{p}) + \mathbf{t}\}$

FAST selects keypoints $\mathbf{p}$ for which either S_t^- or S_t^+ contains n contiguous neighbours in $B_3(\mathbf{p})$. It appears that t is a contrast parameter, whereas n is a geometric (cornerness) parameter.

To avoid selecting adjacent keypoints, FAST restricts the selection to local maxima (over the 3 × 3 neighbourhood) of the following salience function:

$$\Sigma_t^{\mathrm{FAST}}(\mathbf{p}) = \max\left(\sum_{\mathbf{q}\in\mathbf{S}_t^+(\mathbf{p})} \delta_\mathbf{t}(\mathbf{p},\mathbf{q}), \sum_{\mathbf{q}\in\mathbf{S}_t^-(\mathbf{p})} \delta_\mathbf{t}(\mathbf{p},\mathbf{q})\right)$$

with $\delta_t(\mathbf{p},\mathbf{q}) = |\mathbf{I}(\mathbf{q}) - \mathbf{I}(\mathbf{p})| - \mathbf{t}$. FAST is renowned to combine low computational complexity and high repeatability. We will show further that it can also be well suited to our requirement of getting as many points as possible, by adapting the selection strategy.

2.1.2 The MIEL detector

The MIEL (French word for "HONEY") detector is computed on the same support $B_3(\mathbf{p})$ as FAST. Let $\{\mathbf{q_i}\}_{\mathbf{0}\leq\mathbf{i}\leq\mathbf{15}}$ be the 16 points of $B_3(\mathbf{p})$, numbered clockwise. MIEL is based on the following salience function:

$$\Sigma^{\mathrm{MIEL}}(\mathbf{p}) = \min_{\mathbf{i=0}}^{\mathbf{7}} |\mathbf{2I}(\mathbf{p}) - \mathbf{I}(\mathbf{q_i}) - \mathbf{I}(\mathbf{q_{i+8}})|$$

It is based on the hypothesis that matching will be ambiguous for points where there exists at least one direction along which the gray levels vary linearly. The function is eventually equivalent to the minimum absolute value of the 2nd derivative in the eight directions, approximated by the simplest discrete kernel to minimise the number of pixel reads.

The keypoints are then selected using a threshold τ on the salience function. A local maxima strategy like FAST can then be used, but we describe hereunder another selection strategy that better suits our needs.

2.1.3 Keypoints selection strategy

As explained in Sect. 2.1.1, FAST selects local maxima of the computed salience function. It limits the redundancy of salient points by preventing two adjacent points from being selected together. However, this is not ideally suited to track a high number of keypoints, because a point with high salience may be trackable even if it is not a local maximum.

For high-density purposes, we use another selection method that allows to extract more points on salient areas: for each 3 × 3 cell, if more than one pixel are selected, we keep only the one with the highest salience. In the following, we call this strategy blockwise maxima (BM).

Figure 2 compares the two strategies.

2.1.4 Lowering the detector computational cost

From a computational point of view, FAST and MIEL have similar worst-case complexity. But it has been shown [18] that, for specific values of the geometric parameter n, FAST computation can be significantly accelerated by testing subsets of $B_3(\mathbf{p})$, to quickly discard a high proportion of non-keypoints. Nevertheless, the optimisation is specific to each value of n, and we will show in Sect. 2.2 that, for our criteria, the best geometric parameter depends on the targeted number of keypoints.

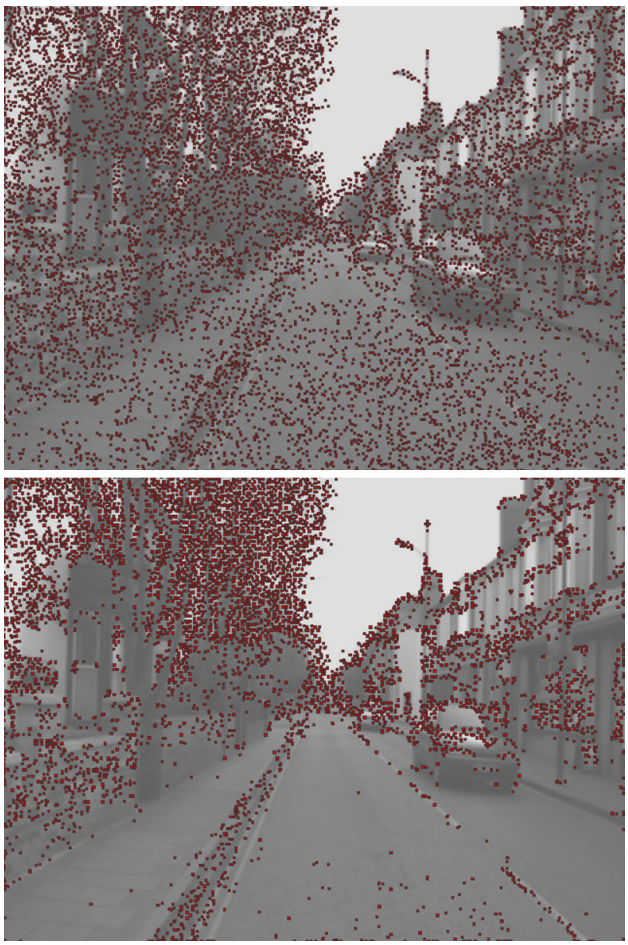

Fig. 2 Comparison of the local maxima (LM) selection (*top image*) and the blockwise maxima (BM) (*bottom image*) using salience. In both cases, 9,000 points are extracted using the MIEL salience (see Sect. 2.1.2). It turns out that, to get a high number of points, the LM method must select unreliable points, whereas our blockwise strategy provides a denser mapping of salient areas

On the contrary, MIEL does not have geometric parameters and lends itself to more straightforward optimisations. As the salience function computes a minimum, it is possible to discard a candidate as soon as a diameter is found along which the second derivative is lower than threshold τ. This dramatically reduces the average number of pixel reads per candidate. For example, two pixel reads are enough to discard points in homogeneous areas.

As opposed to some other tracking algorithms, the matching does not rely on point redetection in each new frame. This limits the use of the detector to new particle detection and thus allows not to run the detector on every frame.

Another straightforward optimisation is to look for new keypoints only on pixels that are not adjacent to existing particles. This reduces the number of pixels the detector has to scan.

2.2 Detector evaluation

This section presents the benchmark used to evaluate the ability of a detector to provide trackable particles. As detailed later, to match one particle **p** from one frame to the other, our method predicts the next position of **p** and uses a gradient descent to refine the matching. Then, errors appear where the good matching is not directly accessible by gradient descent. If the prediction is consistent, this happens when neighbouring points have the same appearance: on homogeneous areas (road, sky,...) or on straight contours (building border, poles,...). The quality of a detector depends on its ability to select the highest number of non-redundant particles with the smallest matching error.

Thus, a detector can be characterised by the sum of matching errors on all selected particles for a given number of particles. Such a number will actually depend on the contrast threshold t (as far as the geometric parameter n of the FAST detector is concerned, different sample values will be evaluated separately hereunder, cf. Fig. 5).

Since the thresholds (t and τ) have different roles in the two detectors, comparing the output of the detectors with the same thresholds does not make sense. Instead, we run the detectors with all possible values of t and τ, and extract for each run the keypoints quality and the number of extracted keypoints. It allows us to compare the detectors in terms of quality with respect to the number of keypoints.

2.2.1 Protocol

The evaluation consists in analysing particles created by the different detectors using the following method. For 100 random translation vectors **v** of norm 5 (as explained below):

- Run the detector on a real-world image.
- Translate the image according to vector **v**.
- Alter image gray levels with a Gaussian noise ($\sigma = 5.0$) to simulate camera artifacts.
- Match the particles extracted at step 1 to find their new position in the transformed image.
- The error on one particle is the distance (in pixels) between its new position and **v**.

Because detectors, even isotropic, behave slightly differently according to orientation, scores are averaged on all images orientations, using a step of 1 degree.

Figure 3 shows the test image with the points extracted by FAST, MIEL and the ideal detector (cf. Sect. 2.2.2). This image has several advantages for evaluating a keypoint detector. First, its size and the amount of information it contains allow to get thousands of keypoints on different kinds of texture. Indeed, this urban scene includes highly textured areas of different natures (buildings, cars, tree,...), non-textured areas like the sky, and the road containing a

Fig. 3 *From top to bottom*: 1—test image of the detector benchmark. 2—Points extracted with an ideal detector based on the matching error map (10k points with the smallest matching error). 3—10k FAST keypoints. 4—10k MIEL keypoints

very high frequency texture. It also contains challenging parts like straight lines, where the matcher often fails. Furthermore, the perspective of the street offers a variety of texture scales: large objects on the foreground, very small objects close to the vanishing point. For all these reasons, this urban scene is well suited to evaluate a keypoint detector and we chose it as input to our benchmark.

As we target video tracking, and thanks to the prior displacement vector estimated for each particle, we assume that the distance between prediction and true match is small. Thus, we limit the displacement of our evaluation to a norm of 5 pixels.

2.2.2 *Results*

In this section, several detectors with different selection strategies are compared according to the previous protocol. They should outperform the random detector, which randomly selects a given number of points, and be outperformed by the ideal selection, which can be applied *a*

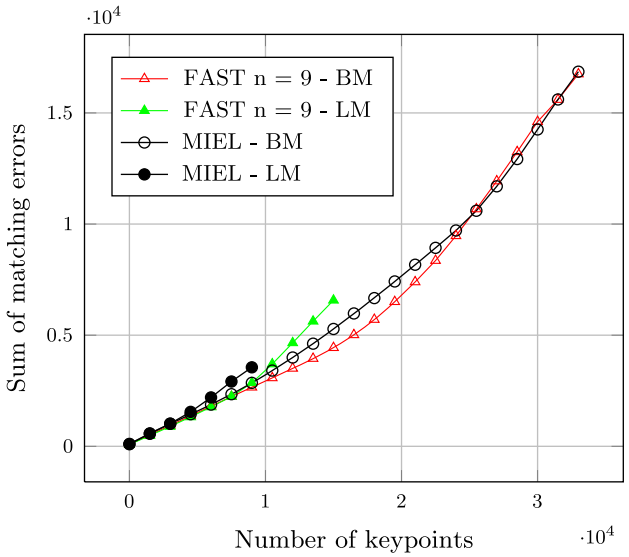

Fig. 4 Selection strategy comparison. Although simpler to compute, blockwise maxima is close or even better (more than 5,000 MIEL or 8,000 FAST points) than the local maxima strategy

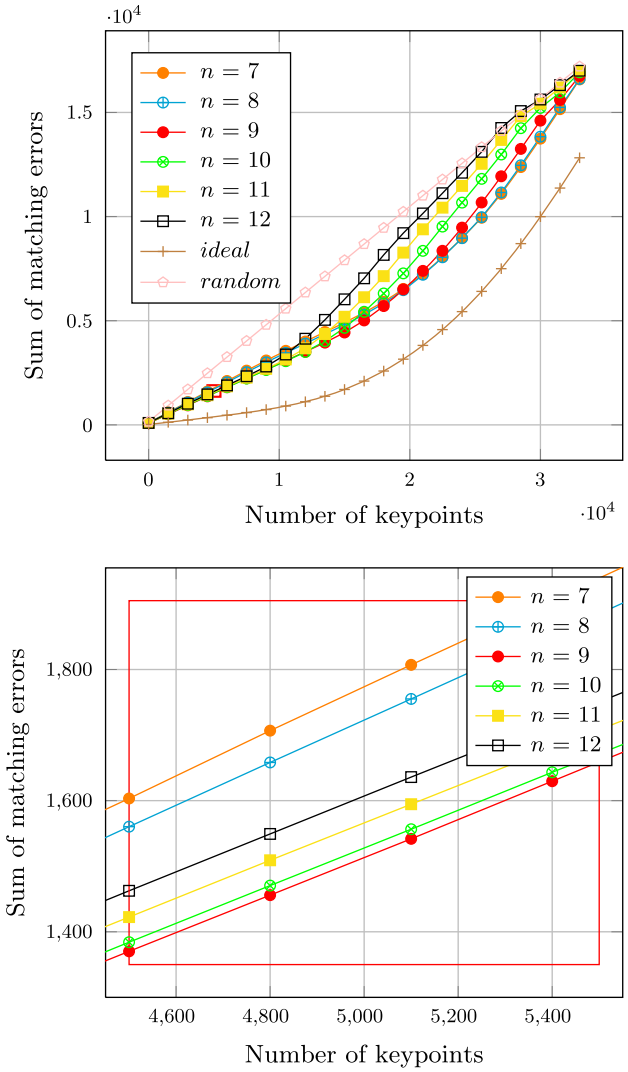

Fig. 5 Profiling the FAST detectors. Best performance is obtained for $n = 9$ below 20,000 extracted points) and $n = 8$ (above)

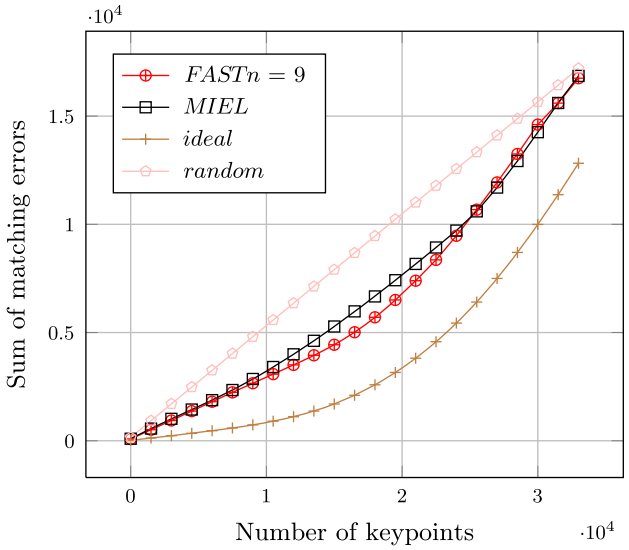

Fig. 6 Comparison of FAST and MIEL. MIEL turns out comparable to FAST tuned at its best, though a little weaker below 25,000 particle

posteriori using the matching error map. The random detector uses a random image as salience function, whereas the ideal detector uses the inverse of the matching error.

Figure 4 shows the impact of our BM selection strategy (cf. Sect. 2.1.3). It allows to extract more points and, above 8,000 FAST points or 5,000 MIEL points, results in smaller matching errors.

Figure 5 compares several versions of the FAST detector. It shows that extraction is optimal with $n = 9$ for less than 20,000 points, whereas $n = 8$ is better for more than 20,000 points. Using our selection strategy, up to 34,133 points are extracted from the 640 × 480 pixels of the test image.

Finally Fig. 6 compares the quality of the best FAST and MIEL keypoints versus the ideal and random detectors. Those results can be outlined as follows:

– Up to a certain limit (around 30,000 particles in our benchmark, that is 10 % of the whole image area), there is a significant benefit in using a geometric selection instead of a random selection.
– The MIEL and FAST detectors have approximately the same quality for less than 10,000 points; it actually seems from our experiments that many different detectors with the same support $[B_3(\mathbf{p})]$ and complexity can achieve similar performance.
– Changing the selection strategy from local maxima to blockwise maxima, as described in Sect. 2.1.3, does improve the detector by allowing a higher density of keypoints.
– The comparison with the ideal detector shows that there is still a large margin of improvement for better detectors.

2.3 Keypoint descriptor

This section introduces a new keypoint descriptor, designed for real-time tracking and specially adapted to the matching algorithm introduced later. Generally speaking, a descriptor has two main characteristics:

– Discrimination power: descriptors should be able to distinguish one image location from a set of matching candidates. This characteristic is related to the size of the descriptor: the larger the descriptor, the more it can distinguish a particle from its neighbours, thus limiting the temporal aliasing problem.
– Invariance: descriptors should be robust to the geometric and photometric changes that may occur from one frame to the other, like viewpoint changes, non-rigid object motions, illumination changes, etc.

Those two characteristics are, to a large extent, antagonist. Besides, high invariance implies a computationally

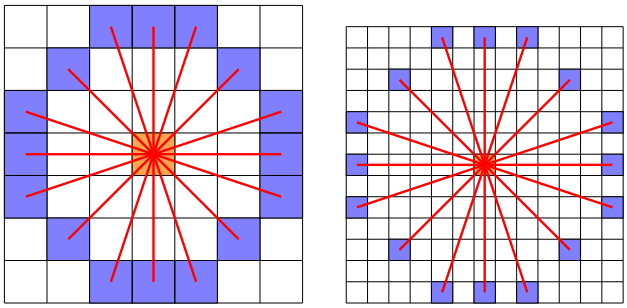

Fig. 7 The two neighbourhoods B_3 and B_6 used to compute the detector and the description vectors

expensive descriptor construction. Our descriptor is justified by the following arguments: (1) using prediction and coarse-to-fine matching, the search space can be reduced, which lowers the importance of discrimination. However, looking for semi-dense particle flow, similar particles can be expected near the search area, so there must be enough bins to distinguish them. (2) Since the target is video tracking, the appearance of objects is not expected to vary significantly between consecutive frames. This lowers the importance of invariance.

Our semi-dense tracker uses a feature vector of dimension 16, corresponding to two sets of eight values evenly sampled on the Bresenham circles of radius 3 (first scale) and 6 (second scale), as shown in Fig. 7. Those values are simply taken from the input image, smoothed by a Gaussian kernel of standard deviation 1.0 for the first scale and 2.0 for the second scale. The matching algorithm presented in the next section uses the $\mathcal{L}_1$ distance between these vectors as a similarity metrics.

This descriptor has two advantages. First, it has a low computational complexity. Second, the use of the Gaussian blur allows to reduce the number of local minima in the distance map, thus improving the accuracy of the matching based on a gradient descent (see Sect. 3.3).

3 Tracking algorithm

This section presents how particle positions are tracked from one frame to the other. It involves prediction, matching, filtering and merging mechanisms.

We first give an overview of the algorithm and then go into the details of each step.

3.1 Overview

We track particles from coarse to fine scales using a dyadic pyramid. Statistics extracted at scale $s+1$ and/or motion estimated at time $t-1$ are used to predict particles at time t and scale s (Sect. 3.2).

The new position of each particle is then estimated by searching its best matching position using a two-scale gradient descent (Sect. 3.3).

To avoid particle drift around occlusions, particles are discarded when the matcher converged to a pixel that is too dissimilar according to the descriptor distance. Error filtering then discards remaining false matches, using spatial statistics (Sect. 3.4).

Finally, to minimise redundant tracking, adjacent particles with similar trajectories are merged (Sect. 3.5).

3.2 Coarse-to-fine prediction

To be robust to large motion due to sudden camera accelerations or fast moving objects, particle positions are predicted using a hybrid temporal and coarse-to-fine spatial method.

Given a particle living at time t and at scale s, let P_t^s be its position in the current frame and $V_t^s = P_t^s - P_{t-1}^s$ be its velocity. To initialise the pyramidal matching framework, at the coarsest scale $s = s_{\max}$, particle positions are predicted to be:

$$\widehat{P_t^{s_{\max}}} = P_{t-1}^{s_{\max}} + V_{t-1}^{s_{\max}} \cdot$$

The position is refined using the matching algorithm (Sect. 3.3), providing $P_t^{s_{\max}}$ and then $V_t^{s_{\max}}$.

Then, particles at finer scales ($s < s_{\max}$) essentially inherit from particles at coarser scales. However, because of sparsity, particles at scale s do not always have corresponding particles at scale $s+1$. The following strategy is used to maximise the probability that a particle inherits a flow vector. For all scales $s < s_{\max}$:

- V_t^{s+1}, the velocity field calculated at the previous (upper) scale, is sub-sampled by replacing the values within every 8×8 block by the average value of the velocities of all particles present in this block. The modified field is denoted $\widetilde{V_t^{s+1}}$.
- The position of each particle is predicted as:

$$\widehat{P_t^s} = P_{t-1}^s + 2\widetilde{V_t^{s+1}}[P_{t-1}^s/2] \cdot$$

- If the corresponding block $\widetilde{V_t^{s+1}}[P_{t-1}^s/2]$ is empty, the position of the particle is predicted as:

$$\widehat{P_t^s} = P_{t-1}^s + V_{t-1}^s \cdot$$

- The matching procedure (Sect. 3.3) is used to refine the particle position P_t^s and velocity V_t^s.

A slight disadvantage of this prediction strategy is that it may induce errors on the borders of moving objects. However, it has two significant advantages:

- It needs less memory to store $\widetilde{V_t^{s+1}}$.
- It smooths matching errors.

3.3 Matching

To refine the position search of each particle, we use a hierarchical sequence of two gradient descents. The following algorithm takes as inputs a point descriptor *F*ref and a predicted new position *p* and returns a refined new position, given a matching distance *d*. Function $F(x)$ computes the descriptor located at position x.

```
descent(reference_descriptor Fref,
        descriptor_function F,
        prediction_position p,
        distance d) {
  new_pred = p
  d_min = d(Fref,F(p))
  best_q = p
  repeat {
    local_min = 1
    for all q neighbour of new_pred {
      if (d(Fref,F(q)) < d_min) {
         d_min = d(Fref,F(q))
         best_q = q
         local_min = 0
        }
    }
    new_pred = best_q
  } until (local_min == 1)
  return best_q
}
```

To improve matching robustness and speed, we use a two-step coarse-to-fine gradient descent. It exploits the two-scale components of our descriptor (see Sect. 2.3). Let $F1$ (resp. $F2$) be the part of the descriptor containing values extracted at the first (resp. second) scale. Let $d1$ (resp $d2$) be the $\mathcal{L}_1$ distance between sub-descriptors $F1$ (resp $F2$).

For a given particle with a descriptor *F*ref and a predicted position p, we estimate its final refined position with the following:

```
descent(Fref, F, descent(Fref, F, p, d2), d1+d2)
```

The first (coarse) descent finds the local minimum according to $d2$, whereas the second (fine) descent uses $d1 + d2$.

Table 1 Comparison of PyrLK [3], using three different integration window sizes (WS), with our tracker

	PyrLK WS = 5	PyrLK WS = 11	PyrLK WS = 21	Our tracker
Scenario S_A— 5,000 particles				
Matching error	1.74	0.92	1.03	1.12
Lost particles	8.27 %	8.26 %	7.91 %	8.82 %
Undetected occlusions	10.80 %	1.41 %	3.39 %	12.82 %
Scenario S_B—5,000 particles				
Matching error	3.62	1.12	1.18	0.94
Lost particles	10.59 %	7.99 %	7.88 %	8.48 %
Undetected occlusions	9.93 %	1.01 %	1.95 %	5.47 %
ScenarioS_A—15,000 particles				
Matching error	2.45	0.99	1.03	1.14
Lost particles	8.92 %	8.39	8.54 %	8.33 %
Undetected occlusions	15.00 %	1.12 %	1.98 %	9.28 %
Scenario S_B—15,000 particles				
Matching error	3.82	1.19	1.19	0.95
Lost particles	12.80 %	8.13 %	8.56 %	8.74 %
Undetected occlusions	8.98 %	0.72 %	1.19 %	4.11 %

Lost particles and undetected occlusions are expressed in percentage of computed trajectories

To handle occlusions, matches are rejected when the similarity distance $d1 + d2$ is above a given threshold θ which sets a balance between robustness to appearance changes and occlusion detection. We used $\theta = 300$ to obtain the results shown in Table 1, which correspond to 7.5 % of the theoretical maximal value of $d1 + d2$.

3.4 Error filtering

Being able to detect false matches is essential to compensate for errors. The latter are mainly due to (1) the limited discrimination power of the detector and (2) the reduced search performed by the gradient descents.

We chose not to perform spatial smoothing but only to remove particles with a spatially inconsistent velocity vector. The false matching detection uses the sub-sampled velocity map presented earlier (Sect. 3.2) to estimate the velocity divergence between the particle and its neighbourhood. Particles verifying the following properties are discarded:

$$||V_t^s - \widetilde{V_t^s}|| > \lambda$$

We use $\lambda = 10$ pixels in our experiments. Furthermore, isolated particles in their 8×8 block are deleted.

Fig. 8 *From left to right*: the input video, the particles (*in red*) and their trajectories (*in black*), and the semi-dense motion field, representing polar coordinates of velocity vectors using an (*intensity, hue*) colour code

Although these two strategies can delete good particles, it has negligible impact on the high number of particles, while improving significantly the average matching error. On the other hand, using statistics already computed for the prediction, this error filtering function has minor computational cost, for the benefit of the overall tracking speed.

3.5 Particle merging strategy

When tracking a semi-dense field of particles, the probability that two particles converge to the same trajectory is high. It can lead to redundant computations, which lowers the performance of the tracker. To avoid this problem, particles which are 1-pixel distant or less are simply merged by removing the youngest. Looking for superposed or adjacent particles can be performed in $O(1)$ time complexity thanks to the spatial index described in Sect. 4.2.

3.6 Evaluation of the tracker

Some results of the semi-dense tracker can be seen on Fig. 8, on different scenarios and scenes. In this section, we propose a method to evaluate our semi-dense tracking algorithm and compare it with others. Comparison is performed with (1) the OpenCV pyramidal Lukas-Kanade (pyrLK) tracker [3] that is widely used as a state-of-the-art algorithm for tracking a high number of points in real time, and (2) Farnebäck's dense optical flow algorithm [8], also available on OpenCV.

The pyrLK tracker does not perform direct search matching, but uses an iterative Euler Lagrange resolution scheme instead. The result quality is related to the smoothness of the function to minimise, which depends on the size of the window used to integrate the derivatives [3]. Since such size has a major influence on the computation time, we use different window sizes for PyrLK on our benchmarks (the default size is 21 pixels in the OpenCV function).

The Farnebäck's dense optical flow [8] is based on polynomial expansion, which estimates coefficients from weighted least squares on the image. The window size of the Gaussian kernel used to calculate the weights also has a strong impact on both quality and robustness to fast motion, so we consider different sizes for Farnebäck too (the default size is 15 pixels in the OpenCV function).

For *Video Extruder*, the smoothness of the similarity function is related to the size and scales of the descriptor. One single version is evaluated, with two scales and eight sample values per scale as described in Sect. 2.3.

3.6.1 Generating sequences with ground truth motion

To benchmark tracking algorithms, we generated synthetic videos with their associated ground truth motion data. The generator produces flat-world dynamic scenarios using a very large image as panoramic background and three small images of objects inserted in the video stream. Two kinds of motion are simulated: camera pan-tilt that affects the whole scene (background and objects), and object-specific motion.

Let α_t be the pan-tilt random acceleration vector at frame t, and β_t^i the random acceleration vector of object i at frame t. Orientations of these acceleration vectors are randomly changed every five frames, while their norms are kept constant.

Two scenarios are used in our evaluation. The first one, with small accelerations, is denoted S_A, with $||\alpha_t|| = 1$ and $||\beta_t^i|| = 2$. The second one, with large accelerations, is denoted S_B, with $||\alpha_t|| = 15$ and $||\beta_t^i|| = 5$ (units in pixels per frame^2).

3.6.2 Qualitative evaluation

A qualitative evaluation is first performed, by comparing the trajectories extracted by the tracking algorithms with the true trajectories obtained from the ground truth sequences.

For each scenario, the tracking algorithms are run on a generated sequence of 100 frames. Each computed trajectory c is compared to the reference trajectory r corresponding to the ground truth trajectory starting at the same position. More precisely, let $T_c = \{\mathbf{p}_\mathbf{s}^\mathbf{c}, \ldots, \mathbf{p}_\mathbf{e}^\mathbf{c}\}$ be a computed trajectory, starting at frame s and ending at frame e. The corresponding reference trajectory is $T_r = \{\mathbf{p}_\mathbf{s}^\mathbf{r}, \ldots, \mathbf{p}_\mathbf{f}^\mathbf{r}\}$, such that $\mathbf{p}_\mathbf{s}^\mathbf{r} = \mathbf{p}_\mathbf{s}^\mathbf{c}$, and f is the final frame of the reference trajectory.

1. The average error along T_c is defined as:

$$\sum_{s \le t \le \min(e,f)} \frac{||\mathbf{p}_\mathbf{t}^\mathbf{c} - \mathbf{p}_\mathbf{t}^\mathbf{r}||}{1 + \min(e,f) - s}$$

2. The particle is considered *lost* if $(f - e) > \eta$.
3. There is an *undetected occlusion* if $(e - f) > \eta$.

We set $\eta = 10$ frames to allow small tracking errors, which represents 0.40 seconds in a 25 Hz input video. For all computed trajectories, we then calculate: (1) the mean average error along the trajectories, which measures the ability to track particle positions without drifting (2) the percentage of lost particles that shows the robustness to motion and appearance change and, (3) the percentage of undetected occlusions. In all these statistics, all trajectories have the same weight, whatever their length.

Table 1 compares the *Video Extruder* with the OpenCV pyrLK tracker [3] parametrised with different window sizes (WS) on the two scenarios S_A and S_B. It shows that, while being significantly faster (see Sect. 4), the global quality of

our tracker is close to that of PyrLK in terms of matching error (1). Measures 2 and 3 are in favour of PyrLK because, unlike our tracker, it does not update point descriptors over time. This allows to detect occlusions more easily, especially when they progressively occlude the particle neighbourhood, at the cost of a weaker robustness to appearance changes.

One counter-intuitive observation is that scenario S_A, while featuring small motion, is actually harder to deal with because it contains slow occlusions that trigger progressive drift of the descriptor leading to matching errors smaller than θ.

3.6.3 Comparison with a dense optical flow

Our approach being hybrid between sparse tracker and dense optical flow, it is also legitimate to compare it to the latter. However, it cannot be done as directly as with a tracker like PyrLK, because the output is different. Thus, we have performed a limited comparison with Farnebäck's optical flow [8], a state-of-the-art dense optical flow algorithm provided by the OpenCV library.

Since it does not provide point trajectory but only optical flow 2D vectors, we cannot compare trajectory quality. Instead, we evaluate the accuracy of its optical flow vectors. Since our ground truth video sequences provide a reference optical flow, the average error per point can be easily calculated. Table 2 shows the average error in pixels for the two scenarios S_A and S_B.

The interest of this comparison is to better evaluate the issues of sparsity *vs* density or semi-density in a real-time context. Globally, those figures are not in favour of [8] since they average the error on all pixels, including homogeneous areas where the errors are the most important. However, our experiments showed that, on average, Farnebäck's algorithm provides better point correspondences than *Video Extruder* on its salient points. It would then make sense to use a dense optical flow as an input to guide the particles, as done in *Particle Video* algorithm [19]. But *Particle Video* is not real time and the computational cost of Farnebäck's algorithm (which would be only a part of a dense particle tracker) is one order of magnitude higher than the semi dense *Video Extruder* (See Table 3).

Table 2 Average error (in pixels) per optical flow vector computed by Farnebäck algorithm [8] with different integration window sizes (WS)

	Farnebäck WS = 5	Farnebäck WS = 15	Farnebäck WS = 25
Scenario S_A	3.25	2.14	1.73
Scenario S_B	5.35	3.46	2.86

4 Implementation

4.1 Parallelism on the CPU and the GPU

In recent years, the growing market of smart phones urged processor designers to increase significantly the efficiency of embedded low power chips. Because high frequency cores are subject to physical limits, chips are made increasingly parallel to raise computational power while reducing energy consumption.

To leverage this trend, our tracker is essentially based on two kinds of highly parallel building blocks:

- Pixel-wise operations involved in the tracking pipeline are convolutions with Gaussian kernels and the keypoint detector. We can split them in as many threads as the number of pixels.
- Particle-wise operations are at the heart of all the other parts of the tracker, i.e. prediction, matching, and error filtering. They use a contiguous buffer of particles (Sect. 4.2), smaller than the input image, thus involving less memory transfers than pixel-wise operations. Because of the high number of particles, it is also worth splitting them into thousands of threads, able to make the most of the GPU parallelism.

So in the GPU implementation, there is one thread per pixel or particle. When targeting the CPU, which is a coarse-grained parallel architecture, we assign to each core a subset of the image, or of the particle buffer. To leverage the CPU cache, adjacent subsets are assigned to consecutive cores, and all memory ranges (input and output buffers) needed by all the CPU threads at a given time must exactly fit the available cache size.

The parallel implementations of the tracker are trivial in that they do not use advanced techniques such as GPU

Table 3 Comparing computation times, averaged on the 1,000 first frames of a 640 × 480 pixels video (CamVid data set)

	PyrLK $WS = 5$	PyrLK $WS = 11$	PyrLK $WS = 21$	Färneback	Our tracker
MPixels/s (frames/s)	31 (101.11)	9.6 (31.25)	2.7 (8.95)	3.4 (11.1)	46.66 (151.9)
µs per particle	1.21	3.50	14.44	0.29 (per pixel)	0.77

PyrLK and our tracker were set to track around 8,500 particles per frame. The detectors were run every five frames. The platform used is a Core i5 2,500k at 3.3 GHz

Table 4 Time performance of our tracker on different architectures (with the detector and filtering steps run every five frames)

Architecture	Resolution	Number of particles	Mpixels/s (frames/s)	Cycles per particle
GPU Geforce GTX 460 1.35GHz	640 × 480	8,500	50 (166)	957
CPU quad-core I5 2500k 3.3GHz	640 × 480	8,500	46 (152)	2,550
ARM dual-core STE U8500 1GHz	320 × 240	3,000	0.84 (11)	30,300
ARM single-core IMX.53 1GHz	720 × 288	2,000	2.07 (10)	50,000

shared memory or explicit use of single instruction multiple data (SIMD) extentions. They simply split the data into chunks that are processed by threads without need of any inter thread synchronisation.

The GPU and CPU implementations of *Video Extruder* are open source and available online: http://www.ensta-paristech.fr/~garrigues/video_extruder.html.

4.2 Particle container

To lower the computation time of both the tracker and the applications using it, we use a particle container which allows to:

- Provide at each frame a contiguous buffer of alive particles.
- Get the particle located on a given pixel in constant time.

The first property is needed to efficiently iterate over all alive particles, and the second one to efficiently look for particles at a given location. We choose to store particle data in a contiguous 1D array P and to reference them in a 2D image R. That is, $R(\mathbf{p}) = i$ if $P(i)$, the particle at index i, is located at pixel $\mathbf{p}$.

Keeping up to date a contiguous buffer of alive particles is a complicated task. Particles appear and die at each frame and P needs to be updated accordingly. Also, dead particle deletion and the subsequent compaction procedure that is needed change particle positions in memory, which makes it difficult to attach data to particles. Thus, we provide a procedure to synchronise attribute data with the particle buffer, optionally saving attributes of dead particles.

This provides third-party applications with a way to attach data to moving particles instead of static pixels. This is useful for object segmentation and tracking or particle's depth estimation.

To better use the processor cache, the container reorders the particle buffer every N frames, in such a way that close particles in the 2D space are also close in memory. To achieve this, a coarse-grained parallel compaction algorithm is used on the CPU and a fined grained parallel version of the Thrust library [11] is used on the GPU.

4.3 Time benchmark

Eventually, the density and matching quality of *Video Extruder* are comparable to those of the OpenCV PyrLK tracker as shown in Sect. 3. But its overriding advantage is speed. We show in this section that it outperforms PyrLK tracker, reaching real-time processing rates even on low power architectures such as ARM systems-on-chip embedded in current middle end smart phones.

Table 3 provides some speed figures and ratios, measured on the Camvid [4] urban driving video data set. The proposed tracking algorithm features speedup factors ranging from ×1.5 to ×17 over PyrLK with different window sizes.

Figure 9 shows the computation time distribution among the different parts of the tracker. The two most expensive tasks are the matching and the two Gaussian blurs per scale needed by the descriptor. All the other parts are negligible because they only involve a few memory fetches per particles and basic arithmetic.

We developed three main implementations of our tracker. Two run on standard desktop hardware, respectively, a GPU and a CPU, and achieve ultra fast processing for high frame rate video sources. The third one targets low power ARM processors and achieves real-time video processing at a lower resolution (see Fig. 10). As far as we know, this is the first implementation able to track such a high number of particles at such a frame rate. Note that, except for the convolutions, none of those implementations makes explicit (i.e. not automatically done by the compiler) use of architecture specific optimisations like SSE, NEON

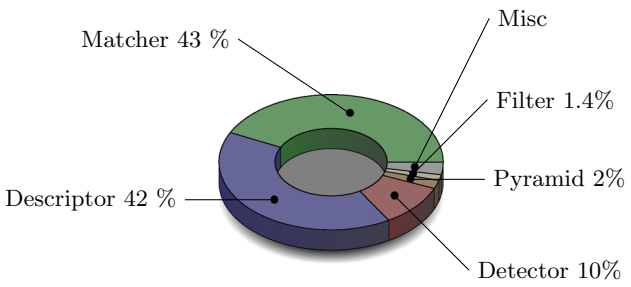

Fig. 9 Processing time distribution among the different parts of the algorithm running on a quad-core x86

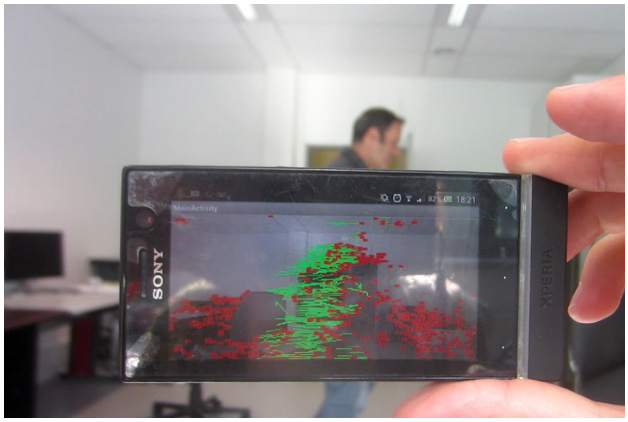

Fig. 10 The tracker running at 10Hz on a low-end Xperia U smart phone embedding an ARM dual-core STE U8500

or other SIMD extensions, so exploiting them could provide further speedup.

Table 4 presents the computation time of our tracker on different architectures, ranging from high performance processors (GPU) to low power ARM processors.

5 Conclusion and perspectives

In this paper, we proposed a visual point tracking algorithm called *Video Extruder*. It is designed to be used as a basic primitive in many embedded video processing systems. Indeed, it lends itself to very fast computation on different architectures, including low power SoCs, thanks to its highly regular and parallel friendly task arrangement and thanks to its efficient data management. Furthermore, it achieves a valuable trade-off between dense optical flow and long-term point tracking. This makes it a versatile brick that can be used in numerous applications: non-rigid object tracking, structure from motion, video stabilisation, video segmentation, action recognition and so on. It has already been used to build action descriptors using beam of trajectories [13, 14]. In [13], it is shown that the performance of action classification is higher when the number of trajectory increases, up to a certain limit: when the quality of matching begins to drop. This confirms the interest of semi dense tracking with respect to sparse tracking and regular/dense sampling. Other contributions of our work are:

- A new salience function which allows fast detection of weakly salient points.
- A new selection mechanism providing much more points than classical local maxima.
- A hybrid temporal and medium range coarse-to-fine scale spatial prediction mechanism which makes the tracker robust to large camera and object accelerations.
- A two scale descriptor with low memory footprint combined with a simple and efficient gradient-descent-based matching algorithm.
- An evaluation protocol adapted to fast mobile video point tracking scenarios.

In our ongoing and future work, we shall use the tracker to build different real-time applications, like software video stabilisation and 3D reconstruction. Besides, the reader is invited to try it for his/her own needs, by downloading the open source version available via the project web page: http://www.ensta-paristech.fr/~garrigues/video_extruder.html.

Other demonstrations and applications are also available on this same page.

Acknowledgments This work was part of a EUREKA-ITEA2 project and was funded by the French Ministry of Economy (General Directorate for Competitiveness, Industry and Services).

References

1. Botella, G., Martín, H.J.A., Santos, M., Meyer-Baese, U.: FPGA-based multimodal embedded sensor system integrating low- and mid-level vision. Sensors **11**(12), 8164–8179 (2011). doi:10.3390/s110808164, URL: http://www.mdpi.com/1424-8220/11/8/8164/
2. Bouchafa, S., Zavidovique, B.: c-velocity: a flow-cumulating uncalibrated approach for 3d plane detection. Int. J. Comput. Vis. **97**(2), 148–166 (2012)
3. Bouguet, J.Y.: Pyramidal implementation of the affine Lucas Kanade feature tracker description of the algorithm. Intel Corporation (2001)
4. Brostow, G.J., Shotton, J., Fauqueur, J., Cipolla, R.: Segmentation and recognition using structure from motion point clouds. In: ECCV (1) pp. 44–57 (2008)
5. Chaudhry, R., Ravichandran, A., Hager, G., Vidal, R.: Histograms of oriented optical flow and Binet–Cauchy kernels on nonlinear dynamical systems for the recognition of human actions. In: Computer Vision and Pattern Recognition, CVPR 2009, IEEE Conference, pp. 1932–1939 (2009)
6. d'Angelo, E., Paratte, J., Puy, G., Vandergheynst, P.: Fast TV-L1 optical flow for interactivity. In: IEEE International Conference on Image Processing (ICIP'11), pp. 1925–1928. Brussels (2011)
7. Doyle, D.D., Jennings, A.L., Black, J.T.: Optical flow background estimation for real-time pan/tilt camera object tracking. Measurement **48**, 195–207 (2014). doi:10.1016/j.measurement.2013.10.025, URL: http://linkinghub.elsevier.com/retrieve/pii/S0263224113005241
8. Farnebäck, G.: Two-frame motion estimation based on polynomial expansion. In: Image Analysis, Springer, pp. 363–370 (2003)
9. Fassold, H., Rosner, J., Schallaeur, P., Bailer, W.: Realtime KLT feature point tracking for high definition video. In: Computer Graphics, Computer Vision and Mathematics (GraVisMa'09), Plzen (2009)
10. Garrigues, M., Manzanera, A.: Real time semi-dense point tracking. In: Campilho, A., Kamel, M. (eds) International Conference on Image Analysis and Recognition (ICIAR 2012), Springer, Aveiro. Lecture Notes in Computer Science, vol. 7324, pp. 245–252 (2012)

11. Hoberock, J., Bell, N.: Thrust: A parallel template library. URL: http://thrust.github.io/, version 1.7.0 (2010)
12. Isard, M., Blake, A.: Condensation—conditional density propagation for visual tracking. Int. J. Comput. Vis. **29**, 5–28 (1998)
13. Nguyen, T., Manzanera, A.: Action recognition using bag of features extracted from a beam of trajectories. In: International Conference on Image Processing (IEEE-ICIP'13), Melbourne (2013)
14. Nguyen, T., Manzanera, A., Garrigues, M.: Motion trend patterns for action modelling and recognition. In: International Conference on Computer Analysis of Images and Patterns (CAIP'13), New York (2013)
15. Rabe, C., Franke, U., Koch, R.: Dense 3D motion field estimation from a moving observer in real time. Smart Mobile In-Veh. Syst. (2014). URL: http://link.springer.com/chapter/10.1007/978-1-4614-9120-0_2
16. Rosten, E., Drummond, T.: Fusing points and lines for high performance tracking. IEEE Int. Conf. Comput. Vis. **2**, 1508–1511 (2005)
17. Rosten, E., Drummond, T.: Machine learning for high-speed corner detection. In: European Conference on Computer Vision (ECCV'06), vol. 1, pp. 430–443 (2006)
18. Rosten, E., Porter, R., Drummond, T.: Faster and better: a machine learning approach to corner detection. IEEE Trans. Pattern Anal. Mach. Intell. **32**, 105–119 (2010)
19. Sand, P., Teller, S: Particle video: long-range motion estimation using point trajectories. In: Computer Vision and Pattern Recognition (CVPR'06), pp. 2195–2202. New York (2006)
20. Schmid, C., Mohr, R., Bauckhage, C.: Evaluation of interest point detectors. Int. J. Comput. Vis. **37**(2), 151–172 (2000)
21. Sekkati, H., Mitiche, A.: Joint optical flow estimation, segmentation, and 3d interpretation with level sets. Comput. Vis. Image Underst. **103**(2), 89–100 (2006)
22. Sinha, S.N., Frahm, J.M., Pollefeys, M., Genc, Y.: GPU-based video feature tracking and matching. In: EDGE, Workshop on Edge Computing Using New Commodity Architectures, vol 278, p 4321 (2006)
23. Sinha, S.N., Frahm, J.M., Pollefeys, M., Genc, Y.: Feature tracking and matching in video using programmable graphics hardware. Mach. Vis. Appl. **22**(1), 207–217 (2007)
24. Tomasi, C., Kanade, T.: Detection and tracking of point features. Carnegie Mellon University Technical, Report CMU-CS-91-132 (1991)
25. Wang, H., Kläser, A., Schmid, C., Chen, g., Lin, L.: Action recognition by dense trajectories. IEEE Conference on Computer Vision and Pattern Recognition, pp. 3169–3176. Colorado Springs (2011)

Matthieu Garrigues got the engineer degree from EPITA. He attended the R&D curriculum at LRDE and made a research period at Siemens CR in Princeton in 2009. He is interested in image processing and embedded implementations for vision systems-on-chip. Since 2010, he has been a research engineer at ENSTA-ParisTech in the Robotics and Computer Vision (RCV) group from the Computer Science and System Engineering (U2IS) Laboratory. He began a PhD thesis in 2012, on parallel vision algorithms for real-time urban scene analysis.

Antoine Manzanera got B.S. degree in Mathematics (1991) and M.S. degree in Theoretical Computer Science (1993) from Lyon I University, Ph.D. from Télécom ParisTech in 2000, and Habilitation degree (HDR) from Paris VI University in 2012. He has been a high school teacher in Paraguay (1994–1996) and a R&D engineer (CIFRE Grant) at Aérospatiale-Missiles (1997–2000). Since 2001, he has been an associate professor at ENSTA-ParisTech in the RCV group from the U2IS laboratory. His research domains are image processing models and algorithms for embedded vision systems, with special interest in motion analysis and understanding.

Thierry M. Bernard got engineer degrees from Ecole Polytechnique in 1983 and ENSTA-ParisTech in 1985. He got M.S. degrees from Paris VI University in 1985, and from CalTech in 1986, and the Ph.D. from Paris XI University in 1992. Formerly with the CTA defence research center, in particular as principal investigator of the Artificial Retina project over 1993–1998 and as scientific manager of the Perception System lab over 1996–1998, he is now an associate professor at ENSTA-ParisTech in the RCV group from the U2IS Laboratory. His research interests are architectures and algorithms for vision systems-on-chip applied to robotics.