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

Matthieu Garrigues, Antoine Manzanera, Bernard Thierry

Noname manuscript No.
(will be inserted by the editor)

Matthieu Garrigues · Antoine Manzanera · Thierry M. Bernard

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

the date of receipt and acceptance should be inserted later

Abstract Two crucial aspects of visual point tracking are addressed in this paper. First, the algorithm should track as many points as possible reliably. Second, the computation should be fast enough, 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 FAST detector. Then, a multi-scale prediction and a matching strategy are presented, based on a 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 in a very fast way. We compare it with the state-of-the-art pyramidal Lucas Kanade point tracker and show that, in fast 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, including 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.

ENSTA-ParisTech,
828 Boulevard des Maréchaux,
91762 Palaiseau CEDEX, France,
http://www.ensta-paristech.fr
E-mail: matthieu.garrigues@ensta-paristech.fr
E-mail: antoine.manzanera@ensta-paristech.fr
E-mail: thierry.bernard@ensta-paristech.fr

1 Introduction

Estimating the apparent displacement of points from a dynamic scene in a video sequence 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 measure 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. For example, this is the case of motion segmentation [16], or when spatial statistics need to be computed on the velocities, e.g. to extract dominant planes by cumulative structure from motion [1], or to recognise actions [4]. Even in applications using long term trajectory estimation (e.g. activity recognition), dense spatial sampling performs better than sparse trajectories of interest points [18].

However there is a certain antagonism between reliability and density and, practically, one has to choose between sparse tracking and 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 point of the video, using short term point matching and spatial regularisation of the motion field. The reasons for 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 implies spatial smoothness.

But this incompatibility is only apparent, since salience or smoothness are not intrinsic to the physical object, but related to its scale of estimation. In practice the distinction is mostly due to 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 answer should

depend on the application, and the choice will clearly limit the versatility of the system.

As an attempt to fill the gap between sparse tracking and dense optical flow, we propose an intermediate approach that performs long term tracking for a set of points (particles), designed to be as dense as possible. Following this approach, we get a flexible motion estimation primitive, that can provide both a beam of trajectories with temporal consistency and a field of displacements with spatial consistency. Our method 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 implementations for sparse point tracking, e.g. Tomasi and Kanade [17], Fassold *et al* [6], and for dense optical flow, e.g. d'Angelo *et al* [5], but these techniques do not provide the level of flexibility we are looking for. Our approach is closer to the *Particle Video* algorithm of Sand and Teller [14] though it is 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, and finally, our method is real-time by design.

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, providing 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, the 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.

The proposed tracking procedure is split into 3 main steps: matching, filtering and new particle detection. Fig.1 shows an overview of the tracking pipeline, whose inputs are the current frame of the video stream and the (multiresolution) 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 image.

The paper is organised as follows. Section 2 first 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 [2]. Section 4 finally discusses implementation issues, and presents time benchmarks obtained with different architectures.

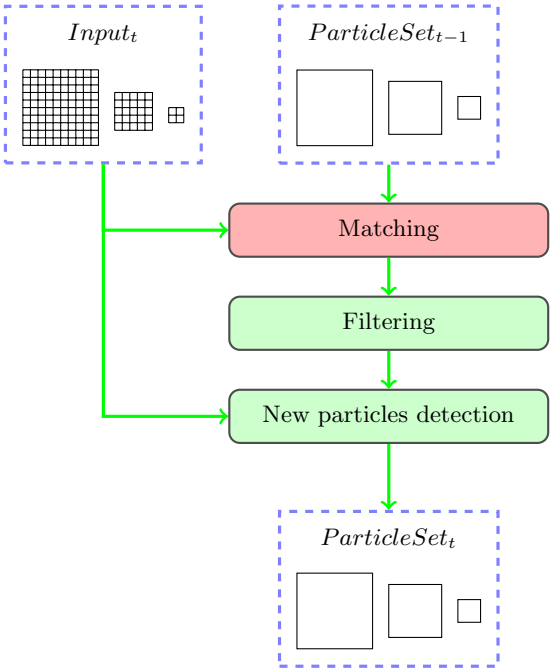

Fig. 1 The tracking pipeline. Only the Matching (in red) has to be run at every frame. The detection and filtering steps may be less frequent. In our applications, they are triggered every 5 frames.

2 Weak keypoint selection and description

The goal of keypoint detection is to select candidate particles in such a way that the largest possible portion of the input video is filled with particles, and only the points whose matching may be ambiguous (homogeneous zones, straight edges) are discarded. We claim that such weak keypoint selection is preferable to dense regular sampling, because it saves a lot of useless computational effort on points whose matching is either very costly or unreliable, often both.

In order to get a particle field that is as dense as possible, the detector needs to be 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 [12] and the MIEL detector, which is an improvement of previous works [7]. Both are computed on the same spatial support B_3 (see Fig. 7),

To assess the different detectors with respect to our requirements, we need to compare (i) the number of keypoints and (ii) the matching errors made by the tracking algorithm on these keypoints. We present our evaluation method and discuss the results in Section 2.2. The repeatability property is usually given much importance [15]. 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 Section 2.3.

2.1 The detectors

2.1.1 The FAST detector

The FAST [11] detector computes local statistics using $B_3(\mathbf{p})$, the Bresenham circle of radius 3 (Fig. 7, left) centred on the keypoint candidate $\mathbf{p}$. Let I be the grayscale image. The detection computes the 2 sets S^- and S^+:

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

FAST selects keypoints $\mathbf{p}$ for which either S_t^- or S_t^+ contains n contiguous neighbours $B_3(\mathbf{p})$. 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:

$$\Sigma_t^{\text{FAST}}(\mathbf{p}) = \max\left(\sum_{\mathbf{q}\in S_t^+(\mathbf{p})} \delta_t(\mathbf{p},\mathbf{q}), \sum_{\mathbf{q}\in S_t^-(\mathbf{p})} \delta_t(\mathbf{p},\mathbf{q})\right)$$

with $\delta_t(\mathbf{p},\mathbf{q}) = |I(\mathbf{q}) - I(\mathbf{p})| - t$. 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 $\{\mathbf{q}_i\}_{0\leq i\leq 15}$ be the 16 points of $B_3(\mathbf{p})$, numbered clockwise. MIEL is based on a salience function which computes the minimal absolute value of the second derivative in the 8 directions:

$$\Sigma^{\text{MIEL}}(\mathbf{p}) = \min_{i=0}^{7} |2I(\mathbf{p}) - I(\mathbf{q}_i) - I(\mathbf{q}_{i+8})|$$

This salience function is based on the hypothesis that matching will be ambiguous for points where there exists at least one direction along which the gray level varies linearly. The function is indeed equivalent to the minimal deviation from linearity in all directions. On areas with salience higher than a given threshold t, the keypoints can then be selected according to a local maxima strategy like FAST, but we describe hereunder another selection strategy that better suits our needs.

2.1.3 Keypoints Selection Strategy

As explained in Section 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 pixels cell of the salience image, the pixel with maximum value is selected if it is greater than the detector threshold. Figure 2 compares the two strategies.

Fig. 2 Comparison of the local maximum (LM) selection (on the top) and the proposed strategy (on the bottom) using salience. In both cases, 9000 points are extracted using the MIEL salience (see Sec. 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.

2.1.4 Lowering the Detector Computational Cost

From a computational point of view, FAST and MIEL have similar complexity. But, it has been shown [13] that, for specific values of the geometric parameter n, FAST computation can be significantly accelerated by testing

subsets from $B_3(\mathbf{p})$, in order to quickly discard a high proportion of the non-keypoints. Nevertheless the optimisation is specific to each value of n and, as will be shown in Sec. 2.2, for our criteria, the best geometric parameter depends on the targeted number of keypoints.

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 the threshold t. This dramatically reduces the average number of pixel reads per candidate. For example, two pixel reads are enough to discard 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 removes the need to run the detector at the same rate as the matcher.

Furthermore, 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, in order to match one particle $\mathbf{p}$ from one frame to the next, our method predicts the next position of $\mathbf{p}$ and uses a gradient descent to refine the matching. Then, errors appear where the good matching is not directly accessible by the 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 particles with the smallest matching error.

Thus, a detector can be characterised by the sum of matching errors on all selected particles, for different number of particles. This number depends on the contrast parameter 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).

2.2.1 Protocol

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

- run the detector on a real world image.
- translate the image according to vector $\mathbf{v}$.
- 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 matched position and $\mathbf{v}$.

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

Figure 3 shows the test image along with the points extracted by FAST, MIEL and the ideal detector (See Sec. **??**). This urban scene is well suited to evaluate a keypoint detector because it features different kind of textures and the two main types of difficulties: straight lines on the road sides and homogeneous areas on the sky and the road.

As we target short range 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 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 made *a posteriori* using the matching error map. The detectors use the same local selection strategy. 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 alternative selection strategy (cf. Sec 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 $30k$ particles in our benchmark, that is 10% of the whole image area), there is a significant benefit in using a geometric selection with respect to a random selection.
- the MIEL and FAST detectors have approximately the same quality for less than 10k points, and it seems from our experiments that many different detectors with the same support ($B_3(\mathbf{p})$) and complexity could achieve similar performance.
- changing the selection strategy from local maxima to blockwise maxima, as described in Sec. 2.1.3, actually improves the detector by allowing a higher density of keypoints.

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.

– 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:

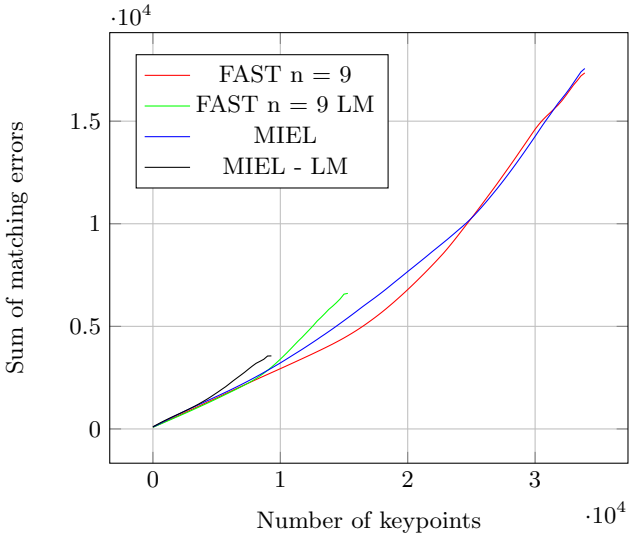

Fig. 4 Comparison of the selections strategies. Although simpler to compute, our selection strategy is close or even better (more than 5 000 MIEL or 8 000 FAST points) than the local maxima (LM) strategy.

– **Discrimination power**: descriptors should be able to distinguish one image location from a set of matching candidates. This characteristics is related to the size of the descriptor: The larger the descriptor is, 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 expensive descriptor construction. Our descriptor is justified by the following arguments: (*i*) by 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. (*ii*) since the target is video tracking, the appearance of objects is not expected to vary significantly between consecutive frames, which limits much the importance of invariance.

Our semi dense tracker uses a feature vector of dimension 16, corresponding to two times 8 values evenly sampled on the Bresenham circles of radius 3 (first scale) and 6 (second scale) (See Fig. 7). First (resp. second) scale values are obtained by smoothing the input frame with a Gaussian kernel with $\sigma = 1.0$ (resp. 2.0). The matching algorithm presented in the next section uses the $\mathcal{L}_1$ distance between these vectors as similarity metrics.

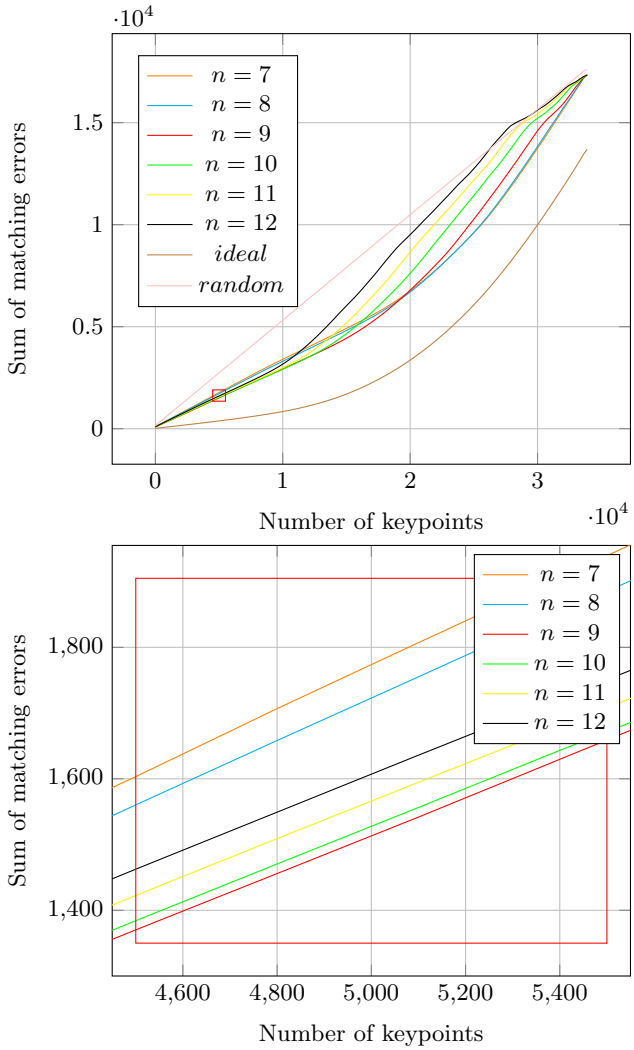

Fig. 5 Profile of the FAST detectors. Best performance is obtained from $n = 9$ (below $20k$ extracted points) and $n = 8$ (above).

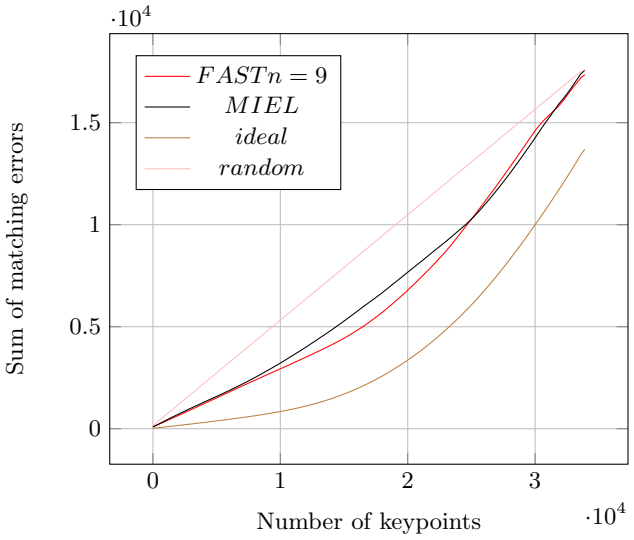

Fig. 6 Comparison of FAST and MIEL. MIEL turns out comparable to FAST tuned at its best, though a little weaker below 25k particle.

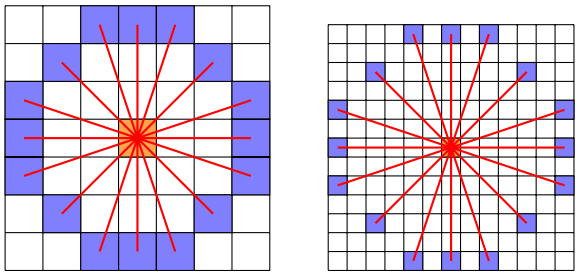

Fig. 7 The two neighbourhoods B_3 and B_6 used to compute the detector and the description vectors.

3 Tracking Algorithm

This section presents how particle positions are tracked from one frame to the next. 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 (Sec. 3.2).

The new position of each particle is then estimated by searching its best matching position using a two-scale gradient descent (Sec. 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 (Sec. 3.4).

Finally, to minimise redundant tracking, adjacent particles with similar trajectories are merged (Sec. 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 its position in the current frame and $V_t^s = P_t^s - P_{t-1}^s$ its velocity. To initialise the pyramidal matching framework, at the coarsest scale $s = s_{\max}$, particle positions are predicted to be:

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

The position is refined using the matching algorithm (Sec. 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:

$$\widehat{P_t^s} = P_{t-1}^s + 2\widetilde{V_t^{s+1}}(P_{t-1}^s/2)$$

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

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

- The matching procedure (Sec. 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 several significant advantages:

- it reduces the amount of memory.
- it smoothes matching errors.

3.3 Matching

To refine the position search of each particle, we use a hierarchical sequence of two gradient descents: Let F_1 (resp. F_2) be the part of the descriptor containing values extracted a the first (resp. second) scale. Let d_1 (resp d_2) be the $\mathcal{L}_1$ distance between sub-descriptors F_1 (resp F_2). The first (coarse) descent finds the local minimum according to d_2, whereas the second (fine) descent uses $d_1 + d_2$. Each step of the two descents looks for minimum distance in a 3×3 neighbourhood.

To handle occlusions, matches are rejected when the similarity distance $d_1 + d_2$ is above a given threshold θ which sets a balance between robustness to appearance changes and occlusion detection. We use $\theta = 300$ to obtain the results shown on Table 1, which correspond to 7.5% of the maximal value of $d_1 + d_2$.

3.4 Error Filtering

Being able to detect false matches is essential to compensate for errors, mainly due to (*i*) the limited discrimination power of the detector and (*ii*) the reduced search performed by the gradient descents.

We chose not to perform spatial smoothing but only remove particles with a spatially inconsistent velocity vector. The false matching detection uses the sub-sampled velocity map presented earlier (Sec. 3.2) to estimate the velocity divergence between the particle and its neighbourhood. We then discard particles such that:

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

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

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, by using statistics already computed for the prediction, this error filtering function has minor computational cost, thus very little influence on 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 merged simply 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 Section 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 the OpenCV pyramidal Lukas-Kanade (pyrLK) tracker [2] that is widely used as state-of-the-art algorithm for tracking a high number of points in real time.

The pyrLK tracker does not perform direct search matching, but uses an iterative Euler Lagrange resolution scheme. The quality of their result is related to the smoothness of the function to minimise, which depends on the size of the window used to integrate the derivatives [2]. Such size has major influence on the computation time, we then consider different window sizes for PyrLK on our benchmarks (the default size is 21 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 2 scales and 8 sample values per scale as described in Sec. 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 3 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 5 frames, while their norms are kept constant.

Two scenarios are used in our evaluation, the first 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^2 per frame).

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 from the same position. More precisely, let $T_c = \{\mathbf{p}_s^c, \ldots, \mathbf{p}_e^c\}$ be a computed trajectory, starting from frame s and ending at frame e. The corresponding reference trajectory is $T_r = \{\mathbf{p}_s^r, \ldots, \mathbf{p}_f^r\}$, such that $\mathbf{p}_s^r = \mathbf{p}_s^c$.

1. The average error along T_c is defined as:

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

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 25Hz input video. For all computed trajectories, we then calculate: (i) the mean average error along the trajectories, which measures the ability to track particle positions without drifting (ii) the percentage of lost particles, that shows the robustness to motion and appearance change and, (iii) 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 [2] parameterised with different window sizes (WS) on the two scenarios S_A and S_B. It shows that, while being significantly faster (see Sec. 4), the global quality of our tracker is close to that of PyrLK in terms of matching error (i). Measures ii and iii are in favour of PyrLK, because, unlike our tracker, it does not update point descriptors over the 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 because it contains slow occlusions that trigger progressive drift of the descriptor components involving matching with error inferior to θ.

4 Implementation

4.1 Parallelism

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 in order 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:

- The **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.
- The **particle-wise operations** affect all the other parts of the tracker, i.e. matching, semi-dense optical flow computation, and error filtering. They use a contiguous buffer of particles (Sec. 4.2), smaller than the input image, thus involving less memory transfers than pixel-wise operations. Because of the high number of particles, they are also easily split 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.

4.2 Particle container

In order 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.

Table 1 Comparison of PyrLK with different integration window sizes (WS) and our tracker. Lost particles and undetected occlusions are expressed in percentage of computed trajectories.

	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 %
Scenario S_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 %

– get the particle located on a given pixel in constant time.

The first property is needed to efficiently iterate over all the 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 particles deletion and the subsequent compaction procedure that is needed change particles position in memory, which makes 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 applications with a way to attach data to moving particles instead of static pixels which is useful for object segmentation and tracking, or particle 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 [8] on the GPU.

4.3 Time benchmark

Eventually, the density and matching quality of Video Extruder is comparable to that of the OpenCV PyrLK tracker as shown in Section 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 2 provides some speed figures and ratios, measured on the Camvid [3] urban driving videos dataset. The proposed tracking algorithm features speedup factors from $\times 1.5$ to $\times 17$ compared to PyrLK with different window size.

Figure 4.3 shows the repartition of the computation time 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. Every other parts are negligible because they only involve few memory fetches per particles and basic arithmetic.

We built three main implementations of our tracker. The first 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. 9). 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 the convolutions, none of those implementations makes explicit (i.e. not automatically done by the compiler) use of architecture specific opti-

Table 2 Compared computation times, averaged on the 1 000 first frames of a 640 × 480 pixels video (CamVid dataset). Both trackers were set to track around 8 500 particles per frame. The detectors were run every 5th frames. The platform used is a Core i5 2500k at 3.3 GHz.

	PyrLK $WS = 5$	PyrLK $WS = 11$	PyrLK $WS = 21$	Our tracker
Frames per second	101.11	31.25	8.95	151.9
μs per particle	1.21	3.50	14.44	0.77

misations like SSE, NEON or other SIMD extensions, so future works could speedup this tracker using such optimisations.

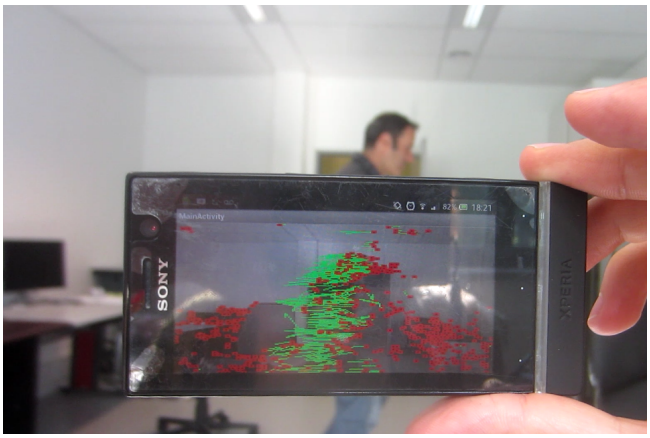

Fig. 9 The tracker running at 10Hz on a low-end Xperia U smart phone embedding an ARM dual-core STE U8500.

Table 3 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 different 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 trade-off between dense optical flow and long term point tracking, which makes it a very versatile brick that can be used in many different applications: non rigid object tracking, structure from motion, video stabilisation, video segmentation, action recognition and so on. It has been used already to build action descriptors using beam of trajectories [10, 9]. In [9], 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 fact confirms the interest of our approach 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 allowing much more points than classical local maxima.
- an hybrid temporal and medium range coarse to 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 gradient-descent based matching algorithm that iterate with a small 3×3 search window.
- an evaluation protocol adapted to fast mobile video point tracking scenarios.

In our on-going 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.

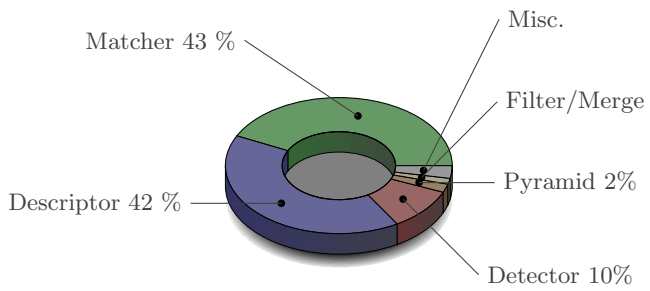

Fig. 10 Repartition of the processing time among the different parts of the algorithm running on a quad-core x86

Acknowledgements

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

Table 3 Time performance of our tracker on different architectures. The detector and filtering step were run every 5 video frames.

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

References

1. Bouchafa S, Zavidovique B (2012) c-velocity: A flow-cumulating uncalibrated approach for 3d plane detection. International Journal of Computer Vision 97(2):148–166
2. Bouguet JY (2001) Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. Intel Corporation
3. Brostow GJ, Shotton J, Fauqueur J, Cipolla R (2008) Segmentation and recognition using structure from motion point clouds. In: ECCV (1), pp 44–57
4. Chaudhry R, Ravichandran A, Hager G, Vidal R (2009) 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, 2009. CVPR 2009. IEEE Conference on, pp 1932–1939
5. d'Angelo E, Paratte J, Puy G, Vandergheynst P (2011) Fast TV-L1 optical flow for interactivity. In: IEEE International Conference on Image Processing (ICIP'11), Brussels, Belgium, pp 1925–1928
6. Fassold H, Rosner J, Schallaeur P, Bailer W (2009) Realtime KLT feature point tracking for high definition video. In: Computer Graphics, Computer Vision and Mathematics (GraVisMa'09), Plzen, Czech Republic
7. Garrigues M, Manzanera A (2012) Real time semi-dense point tracking. In: Campilho A, Kamel M (eds) Int. Conf. on Image Analysis and Recognition (ICIAR 2012), Springer, Aveiro, Portugal, Lecture Notes in Computer Science, vol 7324, pp 245–252
8. Hoberock J, Bell N (2010) Thrust: A parallel template library. URL `http://thrust.github.io/`, version 1.7.0
9. Nguyen T, Manzanera A (2013) Action recognition using bag of features extracted from a beam of trajectories. In: Int. Conf. on Image Processing (IEEE-ICIP'13), Melbourne
10. Nguyen T, Manzanera A, Garrigues M (2013) Motion trend patterns for action modelling and recognition. In: Int. Conf. on Computer Analysis of Images and Patterns (CAIP'13), York
11. Rosten E, Drummond T (2005) Fusing points and lines for high performance tracking. In: IEEE International Conference on Computer Vision, vol 2, pp 1508–1511
12. Rosten E, Drummond T (2006) Machine learning for high-speed corner detection. In: European Conference on Computer Vision (ECCV'06), vol 1, pp 430–443
13. Rosten E, Porter R, Drummond T (2010) Faster and better: A machine learning approach to corner detection. IEEE Trans Pattern Analysis and Machine Intelligence 32:105–119
14. Sand P, Teller S (2006) Particle video: Long-range motion estimation using point trajectories. In: Computer Vision and Pattern Recognition (CVPR'06), New York, pp 2195–2202
15. Schmid C, Mohr R, Bauckhage C (2000) Evaluation of interest point detectors. Int J Comput Vision 37(2):151–172
16. Sekkati H, Mitiche A (2006) Joint optical flow estimation, segmentation, and 3d interpretation with level sets. Computer Vision and Image Understanding 103(2):89 – 100
17. Tomasi C, Kanade T (April 1991) Detection and tracking of point features. Carnegie Mellon University Technical Report CMU-CS-91-132
18. Wang H, Kläser A, Schmid C, Chen g Lin L (2011) Action Recognition by Dense Trajectories. In: IEEE Conference on Computer Vision & Pattern Recognition, Colorado Springs, United States, pp 3169–3176

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