Journal on Real-Time Image Processing manuscript No.
(will be inserted by the editor)

Florence Laguzet · Andres Romero · Michèle Gouiffès · Lionel Lacassagne · Daniel Etiemble

Color tracking with contextual switching

Real-time implementation on CPU.

Received: date / Revised: date

Abstract The paper proposes contributions for Mean-Shift (MS) and Covariance Tracking (CT), and makes these two complementary methods cooperate. While MS runs fast and can handle non-rigid objects represented by their color distribution, CT is more time-consuming but achieves a generic tracking by mixing color and texture information. Each method is modified in order to alleviate their intrinsic limitations, and make the tracking adaptive to a changing context. Concerning MS, the colorspace is changed automatically when necessary to enhance the distinction between the object and the background. Regarding CT, the number of features is reduced without loss of accuracy, by using Local Binary Patterns. Finally, their complementary advantages are exploited in a cooperation process, which runs faster than CT alone, and is more robust than MS alone. A comprehensive study is made for their acceleration and their efficient execution on different multi-core CPU. A speed-up of $\times 2.8$ is reached for MS and $\times 2.6$ for CT

Florence Laguzet
Laboratoire de Recherche en Informatique
University Paris Sud, France
E-mail: florence.laguzet@u-psud.fr

Andres Romero
Laboratoire de Recherche en Informatique
E-mail: andres.romero@u-psud.fr

Michèle Gouiffès
Laboratoire d'Informatique pour la Mécanique et les Sciences de l'Ingénieur
E-mail: michele.gouiffes@u-psud.fr

Lionel Lacassagne
Laboratoire de Recherche en Informatique
E-mail: lionel.lacassagne@lri.fr

Daniel Etiemble
Laboratoire de Recherche en Informatique
E-mail: daniel.etiemble@lrid.fr

1 Introduction

The application field of real-time visual tracking is wide: position control in robotics [26], visual odometry [10], video-surveillance [18] or human-machine interaction [7, 16, 28]. A *good* tracking method requires the balance between several qualities: first it has to run fast and to be robust against common attacks, such as occlusions or illumination variations; second, it has to correctly discriminate the target from its background while tolerating variations of the target itself. In that context, the use of color information is helpful for improving the separability of the target, or defining invariant features [12] to tackle illumination changes. On the downside, it requires a large amount of data to process, making real-time execution more difficult.

Each tracking method has its limitations, and the choice made on the algorithm is closely related to the targeted application, to the nature of the object to be tracked, and also to the possible attacks: occlusion, lighting changes, appearance variation, non-rigid deformation. Some tracking methods have been dedicated for a specific application and use a model of the object to track: the shape of a pedestrian for example, face features [28] or skin color [14]. Unfortunately, in some applications such as video-surveillance [18] it is impossible to have any certainty about the nature of the object, unless a recognition procedure is performed. It is also difficult to forecast the behavior of the tracking from the appearance of the target, only.

Feature points tracking like KLT [6,31] and the extension to object tracking [38] are very popular since points can be extracted in most natural scenes, contrary to many other geometric features. Because they are related to small parts of the image, they are generally considered as planar surfaces and do not undergo large and non-rigid deformations. On the other hand, they are sensitive to occlusions. Although popular detectors/descriptors such as SIFT [25] and SURF [4] can match interest regions undergoing strong scale and rotation changes regardless

of the position, their application is somehow limited to rigid objects, with a sufficient amount of textured patterns.

The kernel-based methods, as Mean-Shift (MS) [9], models the target with a global statistical representation based on color and/or texture, in order to deal with non-rigid motion. However, its good resistance against tricky deformations of the target is counterbalanced by a poor separability with respect to the background. This issue has led to several contributions: comparison of different similarity functions [11], object and/or background classification [33], improvement of the target representation [13] or association with local approaches [2] or corner points [32]. In [23], several histograms from different views are used to improve the modelization of the target.

Covariance Tracking [29, 37] (CT) is an interesting alternative which uses a compact and global representation of the correlation between spatial and statistical features within the object window. High performance can be reached for various kinds of objects, even without texture or feature points. The main weakness of the method is the executing time. Yet, some attempts have been proposed to make it cooperate with particle filtering [24,42], which reduces the tracking time by avoiding exhaustive searching while efficiently estimating the object's state: size, speed, position. The model update can also be improved by using a probabilistic framework [41].

Finally, each tracker cannot suit to every kind of objects or situation. In real-time and mobile video-surveillance applications, different objects are considered (faces, pedestrians, groups, vehicles) in various conditions (outdoor and indoor) at different observation ranges. In addition, the camera can be embedded on a moving vehicle, or can be static for event covering. There are several ways to make several trackers contribute to the process: *competition* [15] where the best result among all methods is picked in each frame, *combination* [1, 2] where the techniques are used jointly to produce the result, and *contextual switching* [8, 14, 21, 40] where either the method or the feature is switched during the tracking only when necessary. For example in [35] and [8], the color representation is chosen automatically during tracking.

Here, in order to reach a good adaptivity of the tracking while keeping real-time execution, the paper: 1) explains our modifications of the tracking methods (Mean-Shift and Covariance) to allows a good adaptiveness to a changing context: illumination changes, background or target appearance variations. In addition, the LBP (Local Binary Patterns) are introduced in the covariance matrix for a better compactness of the descriptor while preserving a good robustness; 2) define a methodology to take benefit of the complementary advantages of the methods, and adapt the tracker to the changing context. In [39], statistical filtering is used with MS to address the problems due to distraction and sudden affine motions of the background, and in [40] CT is performed only when MS fails. The idea of our method is also to combine MS and CT, but this is made differently, in a deterministic way. 3) reduce the executing time on CPU by proper algorithmic modifications: data and structure handling, multi-threading or use of SIMD instructions[1]. This study provides relevant information for implementing the methods for embedded applications. Indeed, a trade-off has to be made between resources utilization and execution times, and multi-threading is not obligatorily the best solution.

The remainder of the paper is structured as follows. Section 2 introduces briefly the two tracking methods. After discussing their contributions and limitations, section 3 proposes a few improvements to better adapt the tracking to the context and finally proposes a methodology to make methods cooperate. The algorithmic modifications to accelerate the execution on CPU, are explained in section 4, and several benchmarks are achieved to prove their efficiency. Experiments on real videos in section 5 confirm the good performance of our MS and CT tracking methods, and proves the good behavior of the cooperation tracking.

2 Discussion on the tracking methods

The target to track is represented by its bounding box $\mathcal{W}$, resulting from a downstream algorithm: for example background subtraction [20] for static cameras, motion analysis [5], recognition and object retrieval methods [36]. Considering a target model in previous frame at time $t-1$, the tracking consists in finding its new location in current frame at time t. This section discusses on the two tracking methods MS and CT that are modified further in section 3.

2.1 Mean-Shift tracking

In MS tracking [9], the target is modeled by a color-space representation, and the tracking is achieved by a gradient-based optimization. The similarity between the target model at initial location $\mathbf{x}_0$ and the target candidate at location $\mathbf{x}_t$, is computed as the Bhattacharyya distance, noted ρ, and based on the bin-to-bin product of their respective color distributions.

If a color appears on both the object and its vicinity or background, it is not relevant for tracking because it reduces their separability. A lower confidence has to be granted to that color, as in [2], where the background colors are subtracted from the histogram using the log-likelihood ratio of foreground/background .

The histogram is generally quantized in order to allow real-time execution and avoid sparsity. Consequently, after background subtraction and quantization, the histogram can be close to empty, and the similarity measure

[1] Single Instruction Multiple Data.

might vanish even for small changes in the color distribution.

Considering the reference target model $\widehat{\mathbf{q}}_u$, the tracking consists in finding in frame t the candidate location $\mathbf{x}_t$ for which the model $\widehat{\mathbf{p}}_u^t(\mathbf{x}_t)$ maximizes the similarity to the model (u is the index on the bins). The Bhattacharyya distance is expanded in Taylor series as in [9] in order to allow gradient based optimization. Iteratively, each pixel of index i and color $\mathbf{c}_i$ in the target contributes to the computation of the new location, with a weight w_i, defined as follows:

$$w_i = \sum_u \sqrt{\frac{\widehat{\mathbf{q}}_u}{\widehat{\mathbf{p}}_u^t(\mathbf{x}_t)}}\delta(\mathbf{c}_i - u) \tag{1}$$

with δ the Kronecker function. Thus, the success of MS relies on the good description of the target by its color distribution. Ideally, the colorspace has to be chosen so that: 1) the histogram is not empty after background subtraction; 2) the weights (1) are non-zero on the target and are numerous enough. Section 3.2 proposes a method to automatically detect the time when the description of the target is no more adapted, and choose the best colorspace to continue the tracking.

2.2 Covariance tracking

In covariance matching, each of n points in the bounding box $\mathcal{W}$, is represented by a vector of d features and noted $\{\mathbf{z}_k\}_{k=1\cdots d}$. Then, the target is described by the compact $d \times d$ covariance matrix $\mathbf{C}_t$. The generic form of the feature vector $\mathbf{z}_k$ can be written as:

$$\mathbf{z}_{\mathbf{A},\mathbf{B}} = \left[\, x \; y \; \mathbf{A} \; \mathbf{B} \,\right] \tag{2}$$

where the spatial coordinates x and y are generally centered and normalized with respect to the object centroid and dimensions. $\mathbf{A}$ is a color feature vector and $\mathbf{B}$ a texture descriptor. Covariance matrices $\mathbf{C}_t$ preserve spatial and statistical information and provides invariance against rotation and scale changes. Tracking an object consists in searching the region of lowest dissimilarity with the object model $\mathbf{C}_{t-1}$ defined in the previous frame using Riemannian geometry [29]. The dissimilarity measure is noted $\mathcal{D}_t$ in the remainder of the paper. Constantly updating the model for changes of shape or color appearance is also necessary. This is done by storing a set of previous covariance matrices $[\mathbf{C}_1 \cdots \mathbf{C}_T]$ where $\mathbf{C}_1$ denotes the current one, and by computing the mean covariance matrix [29].

Contrary to *MS*, the search for new correspondences is generally made exhaustively, except in a few works such as [37]. Exhaustive search is more time-consuming but leads to two advantages: 1) the method is not restricted to small motion, 2) the total occlusions of the object can be tackled by covariance object retrieval [3].

3 Towards a better robustness and adaptiveness to the context

Finally, MS and CT have complementary advantages. MS represents the target by its color distribution, which is almost insensitive to geometric distortions. It implies that the chosen colorspace provides a good distinction between the target and its direct neighboring background, and a good invariance time duration. Section 3.1 details the colorspaces used in the paper, and section 3.2 proposes to automatically choose the colorspace and switch it during tracking, when the context changes. Then, in order to reduce the computation times of CT without robustness loss, section 3.3 explains the features that are used for description. MS runs fast but fails in dealing with large motion and occlusion. Finally, due to a good sepearability power, CT can track most kinds of objects, with large motion, but turns out to be less time effective, as shown further in section 4. The covariance dissimilarity can be an indication to check the accuracy of MS, as made in section 3.4. Then, CT can be run to retrieve objects when lost or occluded.

3.1 Colorspaces and photometric invariance

Improving the robustness against lighting variations can be made by using color invariants such as color ratio, L_1 or L_2 normalization [12] or HSV for example. Several colorspaces are used in the present work: RGB, the normalized composents rgb from the L_1 normalization, as well as several luminance-chrominance representations, such as HSV, IST, YC_bC_r, Yuv and the Otha $I_1I_2I_3$ colorspace[2]. In these representations, the chrominance channels show a low sensitivity to lighting variations. Unfortunately, most chrominance features are ill-defined for low values of saturation or luminance, and can alter the tracking in those situations. The following paragraph proposed a new method to change the colorspace frequently in order to alleviate the issue.

3.2 Adaptiveness to background and appearance changes

As said previously, the success of MS requires the good definition of the weights w_i (1). The number of significant weights (at least positive) has to be large enough so as to provide a reliable approximation of the new location of the target. Second, the colorspace representation has to be chosen so as to maximize the quality of the separability between target and background, and can be updated regularly during the tracking in order to adapt to the context variations. The proposed technique is different from the existing works [8,34]. It provides a good

[2] The reader interested in a further explanation of the colorspaces can refer to [19].

separability target/background, but also tries to ensure the good behavior of MS by checking the consistency of the target model through its weights w_i.

Distinction between target and background. The colorspace has to be chosen so as to maximize the distinction between the histogram of the target h_o and the histogram of its neighboring background h_b, defined as a band of pixels around the target, typically with the same number of pixels as the target. A color u in the target is kept in the model when the following log-likelihood ratio is high enough:

$$L_u = log \frac{\max(h_o(u), \epsilon)}{\max(h_b(u), \epsilon)} \quad (3)$$

where ϵ a small value put to avoid zero at denominator. We note $\mathbf{q}$ the target model, and $\mathbf{p}$ the target model after weighting of each histogram bin u by (3). The colorspace which maximizes ρ between the two histograms $\mathbf{q}$ and $\mathbf{p}$, can be chosen as an ideal colorspace for tracking the considered target. Thus, the following criterion has to be maximized:

$$\mathcal{C}_1 = \rho(\mathbf{q}, \mathbf{p}) \quad (4)$$

However, this is not enough to guarantee a correct and accurate tracking. Imagine that both target and background contain different but close colors. After quantization and background subtraction, some of the colors can be eliminated. As mentioned in section 2.1, it is necessary to take into account the number of the non-zero weights w_i defined in (1).

Amount of weights for the accurate computation of the new location. The previous weighting must not eliminate too many pixels from the target (*i.e* make all w_i vanish). On the other hand, a too large number of positive weights w_i could pervert the tracking when it is due to a *distractor*[3] appearing in the vicinity of the target.

In order to account for these phenomena, the two models are computed as previously $\mathbf{q}$ (without log-likelihood weighting) and $\mathbf{p}$ (with weighting) and the weights w_i are computed by (1). Finally, the criterion of *good colorspace* is given as:

$$\mathcal{C}_2 = \mu_{w_i} = \frac{S_w}{N_w} = \frac{\sum_{\mathcal{W}} w_i}{N_w} \quad (5)$$

where N_w is the number of positive weights in $\mathcal{W}$.

Thus, a colorspace which offers a high separability power has a high value of $\mathcal{C}_1$ in (4) and a high value of $\mathcal{C}_2$, so it can be expressed as a high value of the following criterion C:

$$\mathcal{C} = \mathcal{C}_1\mathcal{C}_2 = \rho(\mathbf{p}, \mathbf{q}) S_w / N_w \quad (6)$$

[3] A distractor is an object, the color distribution of which is approximately similar to the target object.

Switching procedure. Updating the colorspace arbitrarily every K frames can be risky, when it occurs at a bad time, for example when the target is partly occluded or when it brutally changes due to external conditions. To alleviate the problem, the switching decision of figure 1 is proposed. When a scale change occurs[4], the model is updated only when the current ρ becomes lower than the previous one in the previous frame. Otherwise, a procedure is run to decide whether the model has to be updated or not. When the colorspace is changed, the following data are stored: 1) the current distance ρ between the model and the current target; 2) the sum S_w of the corresponding weights w_i in $\mathcal{W}$; 3) the number of positive weights N_w.

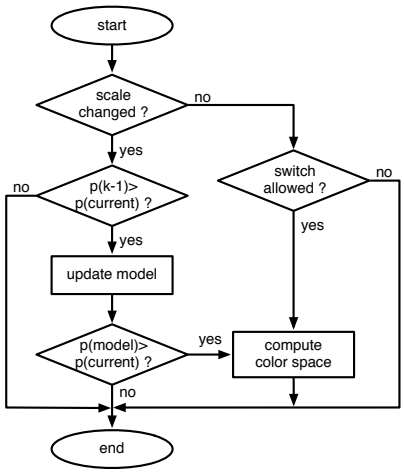


Fig. 1: The colorspace switching.

The colorspace changes when the three indicators ρ, N_w and S_w decrease under 70 % of the value they had at the previous update. In addition, the procedure checks that the ratio N_w/S_w remains under 1, *i.e.* N_w and S_w have to decrease in a similar way. The colorspace which maximizes the criterion $\mathcal{C}$ in (6) is finally selected.

3.3 Robust features in the covariance matrix

In covariance matching, most computational power is spent in comparing the model with candidate covariance matrices. Since the algorithm's total complexity heavily depends on the dimension d, that is $\mathcal{O}(WHd^2 + WHd^3)$ for a target of size $H \times W$, it could be interesting to reduce the number of features without significant loss in terms of robustness.

To that aim, five color sets of features $\mathbf{A}$ are tested: L (a scalar luminance value), $[R\,G\,B]$, $[H\,S\,V]$, $[r\,g\,b]$, and

[4] Let us recall that the scale change is achieved in a similar fashion as [9]

$[L_1\ V]$, as defined in [30] where $L_1 = \max(r, g, b)$ and V is the luminance after photometric normalization. In addition, two texture vectors $\mathbf{B}$ are compared: the gradient vector $grads = [g_x\ g_y]$ and the LBP variance value $\mathcal{V}_S(\mathbf{x}_s)$ computed at location $\mathbf{x}_s$. S uniformly distributed samples are considered in a circular neighborhood of radius R centered on $\mathbf{x}$. The strength of the texture is quantified by the local pixel's variance [27]. Contrary to edges, the operator $\mathcal{V}_S$ is invariant to rotation, and, since it is based on local differences, it is also invariant against intensity shifts. On the other hand, a mere linear edge detector such as Sobel is faster but the two components have to be put in the covariance feature vector.

3.4 Cooperation: handling occlusions and detecting tracking errors

While MS achieves a minimization procedure requiring small inter-frame motion, CT is based on an exhaustive search and is efficient to retrieve objects when lost due to occlusion or false convergence of the minimization. Therefore, when MS fails for some reason – occlusion, bad convergence –, CT is run to retrieve the target. Once retrieved, control returns to MS. The proposed cooperation method is summarized in the flowchart of figure 2. A similar idea has been studied in [40] through a statistical method, where particle filtering is used to handle sudden motion of the camera and distractors appearing in the vicinity of the target. When one of these threats are detected, CT is run to retrieve the target.

The proposed cooperation scheme is different: it is determinist, and no assumption is made on the motion model. Consequently, the compensation of egomotion is not necessary. The failure of MS and CT is detected but it is not necessary to identify the cause, since the method to retrieve the target does not depend on it.

Initialization. First, the detected windows of interest[5] are assumed to contain a target to track. The best colorspace is chosen as detailed in section 3.2, and MS is run. Simultaneously, the covariance matrix of the target C_t is also computed.

Standard tracking. The quality of the MS tracking is checked constantly during the sequence. Unfortunately, for slow drifts of the tracking[6], these criteria can be limiting. The current covariance matrix $\mathbf{C}_t$ and its dissimilarity $\mathcal{D}_t$ with the previous model $\mathbf{C}_{t-1}$ are computed in each frame, in order to confirm the correctness of the tracking, or to prevent from the drift of the target. When $\mathcal{D}_t$ is low, the location of the target is confirmed and the tracking continues with the same method. At each timestamp t, the covariance model $\mathbf{C}_{t-1}$ is updated in order to prepare the next frame-to-frame matching. Here, it is important to underline that the covariance matching is not run, *i.e.* there is no exhaustive search with a large number of dissimilarity measures and comparisons. Here, the target location has already been found by MS, so there is one $d \times d$ matrix to update and one similarity to compute, which is very fast.

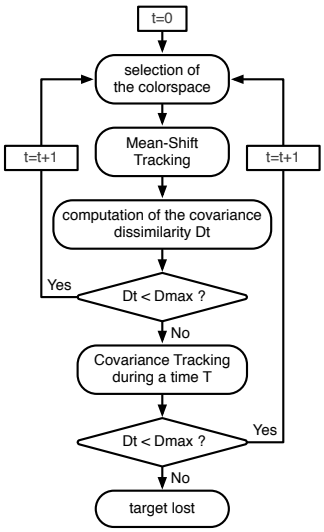


Fig. 2: Flowchart of the contextual switching between MS and CT.

Failure of the tracking. The failure of MS can occur for several reasons: abnormal tracking conditions like occlusions, illumination variations, large motion or appearance changes of the target. Indeed, lighting changes may modify the histogram drastically, unless a specific colorspace is chosen. The deficiency of the tracking can be characterized by a large decrease of the criterion $\mathcal{C}$ in (6). When the tracking begins to loose the target, the covariance dissimilarity $\mathcal{D}_t$ may logically increase as well. Therefore, when $\mathcal{D}_t$ becomes higher than a threshold $\mathcal{D}_{max}$, the covariance matching is run in order to re-acquire the object. When the new $\mathcal{D}_t$ is low enough, *i.e.* target is re-reacquired, the control returns to MS which is less time-consuming, and the tracking becomes standard again. Otherwise, the covariance matching is performed while $\mathcal{D}_t$ remains higher than $\mathcal{D}_{max}$ in order to try to re-detect the target. If the re-detection fails, *i.e.* $\mathcal{D}_t$ remains high after a time T, the target is assumed to be definitely lost.

4 Real-time implementation on CPU

This section explains the techniques implemented to accelerate the tracking algorithms. For sake of conciseness,

[5] The detection problem is evoked in 2. In the paper, we choose to detect the object manually, in order to focus on the tracking problem.

[6] A drift appears when the accuracy of the tracking is reduced during time, and the target is progressively lost.

the explanation is not exhaustive, but concentrates on the three following aspects: the representation of the target, the acceleration of the Mean-Shift and Covariance Tracking respectively.

In order to evaluate the performance of the algorithms and the impact of the optimizations, benchmarks were run on three generations of Intel processors: Penryn/Yorfield, Nehalem/Bloomfield and Sandy-Bridge. Their specifications are collected in Tab.1. All of them are quad-core processors, except Nehalem that is a dual-socket Xeon (2×4 cores). Concerning Sandy-Bridge, the i7-2600K is a processor that can be overclocked (OC). Moreover, thanks to the associated RAM high-speed memory[7], the RAM can be more over-clocked than the processor, which means that, at a higher clock (4.4 GHz) the RAM looks like faster, resulting in smaller transfer times in cycles.

Note that AVX instruction set has not been evaluated, in order to have the same SIMD parallelism for all architectures. Moreover, AVX only includes 256-bit floating point instructions, causing a comparison problem when dealing with integer computations. The quad-core Nehalem and Sandy-Bridge+OC are used to study the acceleration of MS. The third architecture is used for a more thorough evaluation of CT.

4.1 Target representation

The representation of the target is based on several features depending on the method: color (all the methods), histogram (in MS), gradient and covariance matrices (in CT). This section explains our strategies to accelerate each of these components.

4.1.1 Color conversion

As color conversion is used in all the algorithms, this preprocessing step is optimized first. The initial 24-bit images have three components (RGB) per pixel, but as SIMD registers are used, the 24-bit image is converted to 32-bit image by interleaving transparency (*alpha channel*) that will not be used. That makes colorspace conversion much easier by saving a lot of SIMD permutation instructions[8] as in figure 3. Furthermore, some of the colorspaces (Lab for example) require the conversion from 8-bit `unsigned` to 32-bit `float`.

The first step of the optimization is the use of SIMD. It is particularly efficient when some conversions require `if-then-else` tests that are done without pipeline stall. Then, as conversion is pixel-wise, it is embarrassingly parallel, so OpenMP will be efficient (depending on the level of stress on the caches).

[7] It is a 1600 MHz DDR3 with CAS 7, a RAM for *gamer*, nominal DDR3 frequency is 1333 MHz.

[8] Typically `_mm_shuffle_ps` and `_mm_shuffle_epi8` are used

The execution time and the average number of cycles per point (*cpp*) of these versions are presented in table 2 for two quad-core processors: Nehalem and Sandy-Bridge, for an image of size 720×576. The eight colorspaces can be classified according to the computing intensity:

- *lightweight*: RGB is just a copy from a source array to a destination array *without* computation (except cast from/to integer/float). It is memory bound and does not scale (see tab. 2, OpenMP versions)

- *middleweight* (rgb, XYZ, Yuv, $YCbCr$, Otha): these colorspaces are mainly based on a linear combination (3×3 matrix) of RGB.

- *heavyweight* (Lab, HSV): these colorspaces require tests (resulting in pipeline stalls for the scalar version) and complex functions (like trigonometric ones) with a high latency.

We can see that RGB versions are very fast but do not scale due to their *arithmetic intensity* (AI). Concerning the middleweight conversions, they have similar results except rgb which has a bigger *cpp* because it requires a test to avoid zero at denominator. Good speedups are also reached for heavyweight conversions: an average of ×3.4 (efficiency[9] of 85%) and ×3.99 on Sandy Bridge (efficiency of 99.75%). The automatic vectorization has also been tested but has provided poor performance, therefore it is not presented here. However, that enforces the need of handcraft SIMD.

More important than the execution time of each conversion, from a qualitative point of view, our Mean-Shift algorithm needs to compute all the colorspaces to choose the best one to track the target with the maximum robustness. Once the conversions SIMDized, the total execution time drops from 132.99 ms down to 14.41 ms for the Nehalem (7.25 ms for Sandy-Bridge) and when conversions are also parallelized with OpenMP, times decrease to respectively 3.99 ms and 1.90 ms. So, after optimization, there is no trade-off decision to make between speed and robustness, since the quad-core architecture allows to have both.

4.1.2 Mean-Shift optimization

The performance of the MS tracking depend on two points: the quantization of the 3D-histogram and the size of the object to track. As explained in section 2.1, the target is represented by a m-bins histogram. As histograms are central in the algorithm, their dimension affects particularly the execution time. 3D histograms are used in our work. Indeed, 1D or 2D histograms require less resources but do not comprehensively represent the tight relation between color channels. In order to reduce the amount of data to be processed, the image dynamic (initially 256) is quantized linearly with q steps. q has

[9] The efficiency is the speedup divided by the number of cores.

Table 1: Specifications of the processors used in the paper.

Processor	Reference	# cores	GHz	SIMD	RAM (Freq)
Penryn/Yorkfield	Q9550	4	2.8	SSE 4.1	DDR2 PC2-6400 (800 MHz)
Bloomfield	E5530	4	2.4	SSE 4.2	DDR3 PC3-8500 (1066 MHz)
Sandy-Bridge	i7-2600K	4	3.4	AVX	DDR3 PC-10500 (1333 MHz)
Sandy-Bridge+OC	i7-2600K	4	4.4	AVX	DDR3 PC-12800 (1600 MHz)
Nehalem	X5550	2×4	2.66 GHz	SSE 4.2	DDR3 PC3-8500 (1066 MHz)

Table 2: Colorspace: *cpp* and execution time (ms) for quad-core Nehalem and Sandy-Bridge, and for an image of size 720×576.

	Nehalem				Sandy-Bridge			
register	scalar	SIMD	SIMD	SIMD	scalar	SIMD	SIMD	SIMD
threading	mono	mono	OpenMP2	OpenMP4	mono	mono	OpenMP2	OpenMP4
cpp								
Lab	479.78	38.39	20.13	10.48	152.12	31.63	16.23	8.11
HSV	91.06	16.68	8.41	4.38	81.46	14.14	7.09	3.56
rgb	66.97	9.15	4.62	2.41	33.33	8.41	4.14	2.09
XYZ	58.73	7.24	4.51	2.36	36.09	5.96	4.21	2.12
Yuv	58.41	7.95	4.01	2.09	31.86	5.87	2.97	1.49
YCbCr	51.42	6.16	3.10	1.62	24.56	4.48	2.24	1.12
Otha	47.50	6.13	3.43	1.79	23.79	6.88	3.45	1.73
RGB	1.79	1.05	0.80	0.52	1.26	0.73	0.69	0.41
execution time (ms)								
Lab	74.52	5.96	3.13	1.63	14.11	2.93	1.51	0.75
HSV	14.23	2.59	1.31	0.68	7.56	1.31	0.66	0.33
rgb	10.40	1.42	0.72	0.37	3.09	0.78	0.39	0.19
XYZ	9.12	1.13	0.70	0.37	3.35	0.55	0.39	0.20
Yuv	9.07	1.23	0.62	0.32	2.96	0.54	0.28	0.14
YCbCr	7.99	0.96	0.48	0.25	2.28	0.42	0.21	0.10
Otha	7.38	0.95	0.53	0.28	2.21	0.64	0.32	0.16
RGB	0.28	0.16	0.12	0.08	0.12	0.07	0.06	0.04
total	132.99	**14.41**	7.61	3.99	35.67	**7.25**	3.81	1.91

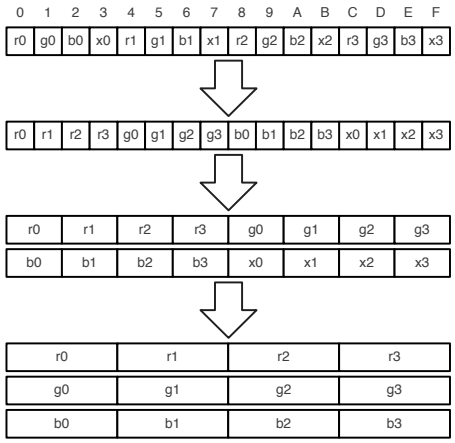


Fig. 3: Storing of color components for SIMD. A 128-bit SIMD register is loaded from memory filled with 8-bit RGBX pixels (first row), is de-interlaced (second row) and converted into two 16-bit vectors of RGBX data (third row) and finally into three 32-bit vectors of *RGB* data (fourth row).

to be chosen depending on a trade-off between accuracy and execution time. Three values of q have been tested: $q \in \{16, 8, 4\}$ leading to three different size of histogram: 16^3, 32^3 and 64^3.

To build this model, several operations are required: quantization, computation of the kernels and log-likelihood for background subtraction. Before analyzing each function, benchmarks for two target sizes were done: small size (28×52 pixels) and big size (99×127 pixels), where each pixel is a color triplet.

In order to determine the new target location, a gradient-based optimization is made using the minimization of the Battacharryya coefficient. As the computation of the new position depends only on computed candidate histogram, we propose a task-parallelism scheme with two threads, as described in figure 4.

In order to evaluate the relevance of this method, the execution time in percentage of each iteration operation is given in table 3. The most significant gains are reached for 32-bins histogram. As execution time depends on the number of optimization iterations, the gains obtained do not fill the extra-cost of using 32-bins histograms instead of 16 bins. As an example, one iteration is almost three times slower with 32 rather than 16.

As previously noticed, three sizes of histogram have been evaluated: 16^3, 32^3 and 64^3. If the execution time of the histogram computation is multiplied by a factor 2^3, when increasing the size of the histogram of a factor 2, this is different for one iteration of the whole MS algorithm. For example, the time is multiplied by $\times 3.5$ for small objects and $\times 1.4$ for big ones, when switching from 16^3 to 32^3 histogram size. From a qualitative point of view, 32^3 histograms allow a better robustness of the tracking compared to 16^3 ones. Then, using a histogram of size 64^3 is not much better than 32^3. Therefore, in order to have both robustness and low execution time, 32^3-size histogram have been chosen for the remainder of the paper.

As shown in table 4, SIMD provides a significant acceleration of $\times 2.7$ in average, while OpenMP is inefficient. Indeed, even if many parts of the algorithm can be multi-threaded, there are sometimes not enough computations to make it efficient, or synchronization barriers (OpenMP) make the computation load not well-balanced. As OpenMP is not efficient, especially for small targets, and since additional algorithms (for example detection or classification) might be implemented to address a comprehensive application, it is preferred to avoid multi-threading in order to let a core available for other computations.

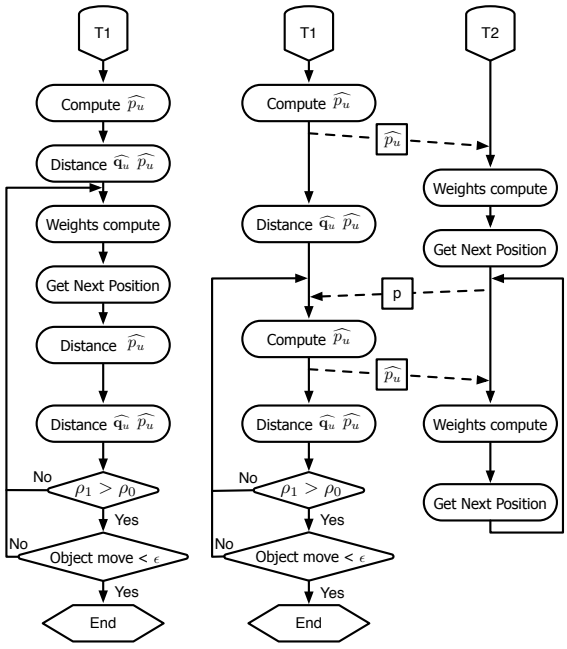


Fig. 4: Mean-Shift gradient-based optimization, *mono-threaded* on the left or *dual-threaded* on the right.

Table 3: Execution time (in percentage) of each function involved in gradient optimization of MS.

	Width of the 3D histogram		
	16	32	64
Histogram Computation	37.27	50.78	57.40
Histogram Distance	10.49	30.03	38.87
Weight Computation	40.14	14.93	3.04
New Position	12.07	4.26	0.69
Item motion	0.04	0.01	0.00

Table 4: Execution time (μs) for MS algorithm and impact of the optimization.

	Nehalem		SandyBridge +OC	
	small object	big object	small object	big object
execution time (μs)				
scalar	714	2933	717	2555
SIMD	280	1042	292	897
SIMD+OpenMP=2	301	778	288	674
SIMD+OpenMP=4	297	719	283	556
impact of optimizations				
SIMD/scalar	$\times 2.54$	$\times 2.81$	$\times 2.46$	$\times 2.85$
mono/OpenMP=2	$\times 0.93$	$\times 1.34$	$\times 1.01$	$\times 1.33$
mono/OpenMP=4	$\times 1.01$	$\times 1.08$	$\times 1.02$	$\times 1.21$

4.2 Covariance Tracking optimization

Two strategies can be used to optimize CT. The first one consists in performing multi-threading by parallelizing the most outer loop. This is done with OpenMP. The second one is based on SoA→AoS (*Structure of Arrays* to *Array of Structures*) transformation. Section 4.2.1 describes the second transform only, the first one being straight-forward. Then, a micro-benchmark of CT is presented in 4.2.2. Finally, section 4.2.3 studies the impact of the proposed optimization on the whole execution time.

4.2.1 SoA→AoS

The goal of *SoA→AoS* manipulation consists in transforming a set of independent arrays into one array, where each cell is a structure combining the elements of each independent array. The contribution of such a transform is to leverage the cache performance by enforcing spatial and temporal cache locality. Let us define the following notations:

- h and w: height and width of the image
- n_F: number of features,
- n_P: number of products of features, $n_P = n_F(n_F + 1)/2$,
- F: a cube (SoA) or matrix (AoS) of features,
- P: a cube (SoA) or matrix (AoS) of feature products,
- I_F and I_P: two cubes (or matrices) of integral images (from F or P).

Here we want to optimize the locality of the features (or the product of features) of a given point of coordinates (i, j). The *SoA* version requires two cubes: F_{SoA} and P_{SoA} respectively of size $n_F \times h \times w$ and $n_P \times h \times w$. The *AoS* is based on two matrices F_{AoS} and P_{AoS} of size $h \times (w \cdot n_F)$ and $h \times (w \cdot n_P)$. Here, the *SoA*→*AoS* transform consists in permutting the loop nests and changing the addressing computations from a 3D-form like $cube[k][i][j]$ into a 2D-form like $matrix[i][j \times n+k]$, where n is the structure cardinal (here n_F or n_P).

The covariance tracking algorithm is composed of three stages:

1. point-to-point products computation of all features,
2. integral image computation of features,
3. integral image computation of products,

The product of features and its transformation are described in algorithms 1 and 2. Thanks to commutativity of the multiplication, only half of the products have to be computed (the loop on k_2 starts at k_1, line 3). As the two last stages are similar, we only present a generic version of integral image computation (Algo. 3) and its transformation (Algo. 4).

Algorithm 1: Optimization of CT - product of features *SoA* version.

```
1 k ← 0
2 foreach k1 ∈ [0..nF − 1] do
3     foreach k2 ∈ [k1..nF − 1] do
4         [ point-to-point multiplication ]
5         foreach i ∈ [0..h − 1] do
6             foreach j ∈ [0..w − 1] do
7                 P[k][i][j] ← F[k1][i][j] × F[k2][i][j]
8                 k ← k + 1
```

Algorithm 2: Optimization of CT - product of features,*AoS* version.

```
1 foreach i ∈ [0..h − 1] do
2     foreach j ∈ [0..w − 1] do
3         k ← 0
4         foreach k1 ∈ [0..nF − 1] do
5             foreach k2 ∈ [k1..nF − 1] do
6                 P[i][j × nP + k] ←
                  F[i][j × nP + k] × F[i][j × nP + k]
7                 k ← k + 1
```

Once this transform is done, one can also apply SIMD to the different parts of the algorithm. For the product part, the two internal loops on k_1 and k_2 are fully unrolled in order to show the list of all multiplications and the list of vectors to construct through permutation instructions[10]. For example, for a typical value of $n_F = 7$, there are $n_P = 28$ products, made in the following order (the numbers in brackets are the feature indexes):

$$[0,0,0,0] \times [0,1,2,3]; \quad [0,0,0,1] \times [4,5,6,1];$$
$$[1,1,1,1] \times [2,3,4,5]; \quad [1,2,2,2] \times [6,2,3,4];$$
$$[2,2,3,3] \times [5,6,3,4]; \quad [3,3,4,4] \times [5,6,4,5];$$
$$[4,5,5,6] \times [6,5,6,6].$$

In that case, the 7th vector is 100% filled, but it will become sub-optimal if n_P is not divisible by the cardinal of the vector (4 with SSE, 8 with AVX). The permutations are made in one instruction for some of them, and in two instructions in the worst case. Because some permutations can be re-used for other permutations, a factorization is made over all the required permutations. For example with $n_F = 7$, fifteen shuffles are required.

Algorithm 3: Optimization of CT - integral image *SoA* version.

```
1 foreach k ∈ [0..n − 1] do
2     [classical in place integral image]
3     foreach i ∈ [0..h − 1] do
4         foreach j ∈ [0..w − 1] do
5             I[k][i][j] ← I[k][i][j] + I[k][i][j − 1] +
              I[k][i − 1][j] − I[k][i − 1][j − 1]
```

Algorithm 4: Optimization of CT - integral image *AoS* version.

```
1 foreach i ∈ [0..h − 1] do
2     foreach j ∈ [0..w − 1] do
3         foreach k ∈ [0..n − 1] do
4             I[i][j × n + k] ←
              I[i][j × n + k] + I[i][(j − 1) × n + k] + I[i −
              1][j × n + k] − I[k][i − 1][(j − 1) × n + k]
```

4.2.2 Micro benchmarks and analysis

The amount of memory required by CT is equal to $(n_F + n_P) \times$ `sizeof(float)` $\times h \times w$ bytes. Assuming $h = w = 1024$, $n_F = 7$ and $n_P = 28$, the algorithm needs 140 MB, much more that the size of the biggest available cache ! In order to evaluate the impact of the optimizations, the three versions of the algorithm (SoA, AoS, AoS+SIMD) have been benchmarked for sizes varying from 128×128 to 1024×1024. The *cpp* results are displayed in table 5 for the different architectures and the gains are collected in table 6. Finally, a very big target is considered in the results of table 7.

Global Observation: with the introduction of each new family (Penryn, Bloomfield/Nehalem, Sandy-Bridge), the

[10] This is made with `_mm_shuffle_ps` in SSE

Table 5: Optimization of CT: *cpp* for 128×128 and 1024×1024 image with 7 features.

	mono-threading			multi-threading		
	SoA	AoS	AoS+SIMD	SoA	AoS	AoS+SIMD
cpp for 128 x 128 images with 7 features						
Penryn	527.5	288.7	103.2	335.4	248.6	95.8
Bloomfield	481.1	255.8	108.7	195.9	249.1	121.6
SandyBridge	351.3	177.1	71.8	154.1	150.0	64.8
SandyBridge+OC	310.3	156.5	62.9	130.7	129.1	54.0
Nehalem	218.0	179.6	78.1	51.8	143.9	62.9
cpp for 1024 x 1024 images with 7 features						
Penryn	586.4	385.9	314.3	366.7	373.8	295.5
Bloomfield	502.0	270.5	150.0	208.3	266.7	160.0
SandyBridge	375.6	190,1	112,1	157.8	185.2	112.2
SandyBridge+OC	326.2	172.2	108.0	140.9	169.9	108.6
Nehalem	263.2	202.3	123.2	75.5	161.3	96.3
slow-down factor between 128x128 and 1024x1024						
min	1.04	1.06	1.38	1.02	1.07	1.32
max	1.39	1.34	3.05	1.84	1.50	3.08

Table 6: Optimization of CT: gain for 128×128 and 1024×1024 image with 7 features.

	mono-threading			multi-threading	
	SoA/AoS	SIMD	total	OpenMP	total
cpp for 128 x 128 images with 7 features					
Penryn	×1.83	×2.80	×5.11	×1.57	×5.51
Bloomfield	×1.88	×2.35	×4.43	×1.76	×7.21
SandyBridge	×1.98	×2.47	×4.89	×2.28	×5.42
SandyBridge+OC	×1.98	×2.49	×4.93	×2.37	×5.75
Nehalem	×1.21	×2.30	×2.79	×4.21	×3.47
cpp for 1024 x 1024 images with 7 features					
Penryn	×1.52	×1.23	×1.87	×1.60	×1.98
Bloomfield	×1.86	×1.80	×3.35	×1.74	×4.03
SandyBridge	×1.98	×1.70	×3.35	×2.38	×3.35
SandyBridge+OC	×1.89	×1.59	×3.02	×2.32	×3.00
Nehalem	×1.30	×1.64	×2.14	×3.49	×2.73

Table 7: Optimization of CT: execution time (ms) for 1024×1024 image with 7 features.

	mono-threading			multi-threading		
Processor	SoA	AoS	AoS+SIMD	SoA	AoS	AoS+SIMD
Penryn	219.6	144.5	117.7	137.3	140.0	110.7
Bloomfield	219.3	118.2	65.5	91.0	116.5	69.9
SandyBridge	103.6	52.5	30.9	43.5	51.1	31.0
SandyBridge+OC	77.7	41.0	25.7	33.6	40.5	25.9
Nehalem	103.8	79.7	48.6	29.8	63.6	38.0

performance of CT increases for all implementations. Indeed, the evolution of the processor architecture – cache and RAM connexion – makes the *cpp* (for SoA version) decrease from 527.5 for Penryn down to 310.3 for SandyBridge (see Tab. 5). In the remainder, the Nehalem will be analyzed separately since: 1) it is the only dual-processor machine of the benchmarks; 2) as a server machine, it has faster buses.

AoS & SIMD: the $SoA \rightarrow AoS$ transform is very efficient: nearly a speedup of ×2. The SIMD version of AoS provides an average speedup of ×2.5 for 128 × 128 images (see Tab.6) due to the small *arithmetic intensity* (AI) of CT. For the product part of the algorithm (Algo. 1) the AI is equal to 0.33 in scalar and 0.77 in SIMD [11]. For the integral-image part (Algo. 3) the AI is equal to 0.6 for both scalar and SIMD versions[12]. When the data do not fit in the cache – for 1024 × 1024 images – this speedup drops down to an average value of ×1.6. When combined together, the speedup reaches a maximum of ×5.11 in

[11] The scalar version requires 1 MUL, 2 LOAD, 1 STORE and the SIMD version: 7 MUL, 15 SHUFFLE, 2 LOAD, 7 STORE.

[12] 3 ADD/SUB, 4 LOAD, 1 STORE.

128 × 128 and ×3.35 for 1024 × 1024. Concerning the octo-core Nehalem, speedups are smaller but execution time (Tab. 7) is smaller too, due to its faster cache-RAM buses.

OpenMP: the gain due to OpenMP is lower – except for Nehalem – compared to the gain due to SIMD with an average value of ×2.0 for 128 × 128 and for 1024 × 1024 images. It can be explained by memory accesses with very large strides that are problematic with optimal cache running. For Nehalem, OpenMP provides speedups of ×4.21 and ×3.49 thanks to the eight cores and caches. But the efficiency remains low: about 0.5.

SIMD+OpenMP: if we combined together AoS+SIMD with OpenMP, speedups are a bit better but efficiency is worst !

Execution time: when focusing on the execution time, one can notice that *SoA*→*AoS* transform and SIMD are mandatory to achieve real-time processing (40 ms per camera frame) for 1024 × 1024. For the mono-threaded implementation, only the SandyBridge (with SSE instructions) is real-time.

Sandy-Bridge+OC: as explained previously, the RAM can be more over-clocked. It results in a smaller *cpp* with a higher frequency. The speedup (between normal and OC) is ×1.33 for *SoA* version and ×1.20 for *AoS*+SIMD version. By combining all together optimization and over-clocking, a speedup of ×4.0 is finally reached. Most importantly, this test shows that memory is still the bottleneck, even when performing memory transformations (*SoA*→*AoS*) for better cache locality.

Conclusion: Here also, when a set of algorithms have to run simultaneously, the best choice is to implement a mono-threaded version of the covariance matching– with *SoA*→*AoS* transform and SIMD – in order to leave the other processor cores free for the other algorithms.

4.2.3 Macro benchmarks and analysis

CT has been benchmarked with sequences used in [30] [13], in monochrome and color versions. Because of the poor performance of the multi-threading versions, the whole algorithm is finally implemented and evaluated in a single-threaded version. The combined impact of the *SoA*→*AoS* transform and the SIMDzation in SSE is evaluated. The measures show that execution times are in the range [3ms .. 11ms] making the algorithm fast and compatible with real-time execution on one core. The corresponding accelerations range between ×1.53 and ×2.60, which is close to the maximum speedup of ×3.35 for the micro-benchmark.

4.3 Manycore implementation on GPU ?

Concerning the opportunity of a GPU implementation, let us discuss on the two most time-consuming parts of our application. The first one is color conversion: *middleweight* and *heavyweight* colorspace conversions have a good scalability with OpenMP. These computations would probably scale on GPU (now from 512 up to 2200 cores) as well. The second one is mean-shift and covariance tracking algorithms. For MS, the biggest speedup is ×1.21 with OpenMP=4, that is an efficiency of 30%. For covariance tracking the efficiencies (repectively for 4-core and 8-core processors) are 60% (×2.38/4) and 53% (×4.21/8). These algorithms will definitively not scale on a manycore processor such as a GPU in their *mono target tracking version*. The only possibility is to boost colorspace conversions on GPU, but, as the next computation will held on GPP, one has to pay the host-to-gpu and gpu-to-host transfers. The transfer duration (compared to the duration of the computation) will make the colorspace acceleration on GPU not efficient.

So the final possibility is to leverage the load of a many-core architecture is doing *many-target tracking*, that is to track from hundreds to thousands of targets (like the particular filter for example). That does not make sense for such a kind of tracking algorithm, where only *multi-target tracking* (from 10 to 50, maybe 100) can be envisionned. What we can naively state is that – in order to get a 100%-loaded architecture – the number of targets to track should be greater or equal to the number of available cores. But performance will depend of the memory-access conflicts (to the global memory or to the caches).

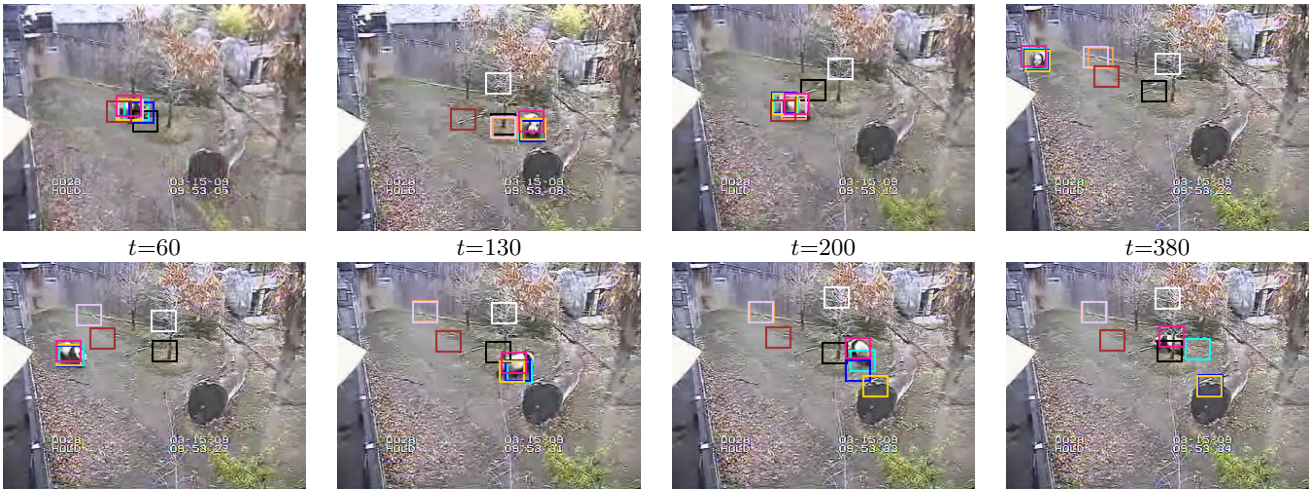
4.4 Benchmark conclusion

Thus, in order to be efficient, it is preferred to implement single-threaded version of MS and CT algorithms. Almost all the internal computation parts of these algorithms can be accelerated with SIMD instructions, and the *SoA*→*AoS* transform turns out to be very efficient for CT algorithm. Then, the use of OpenMP is of high interest for accelerating the color conversions which are finally the most time-consuming functions of the whole application. Finally, due to the important accelerations that have been made, MS, CT and seven color conversions can be executed simultaneously. Note however that CT remains more time-consuming than MS. Considering an embedded platform – a quad-core ARM Cortex architecture – such a choice makes sense.

5 Experiments

This section illustrates the behavior of the tracking procedure: the impact of the colorspace switching on the

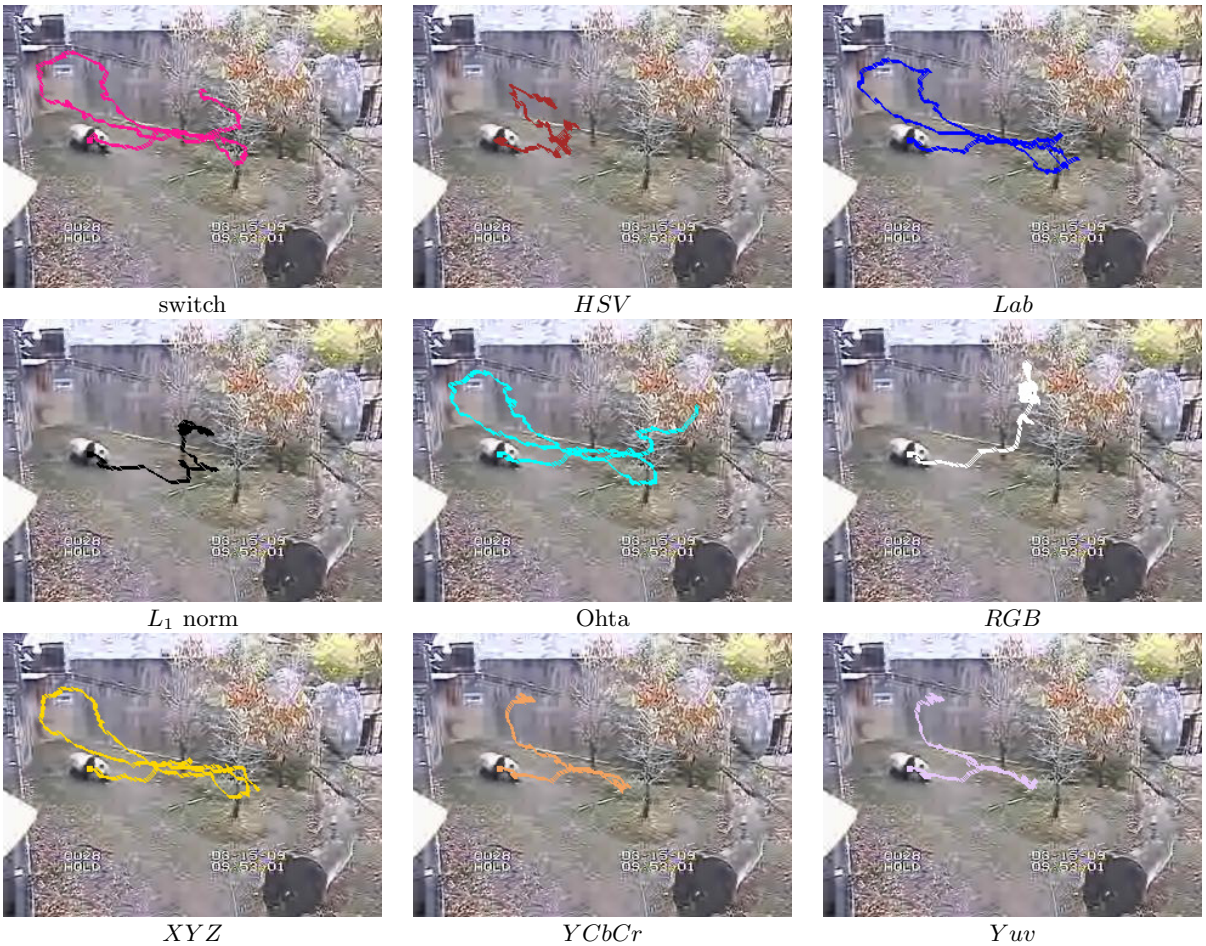
Fig. 6: Color chart used for the results.



t=60 t=130 t=200 t=380

t=470 t=560 t=590 t=610

Fig. 7: Evaluation of MS with colorspace switching on the sequence *Panda*.



switch HSV Lab

L_1 norm Ohta RGB

XYZ $YCbCr$ Yuv

Fig. 8: Trajectories of the tracking methods on the sequence *Panda*.

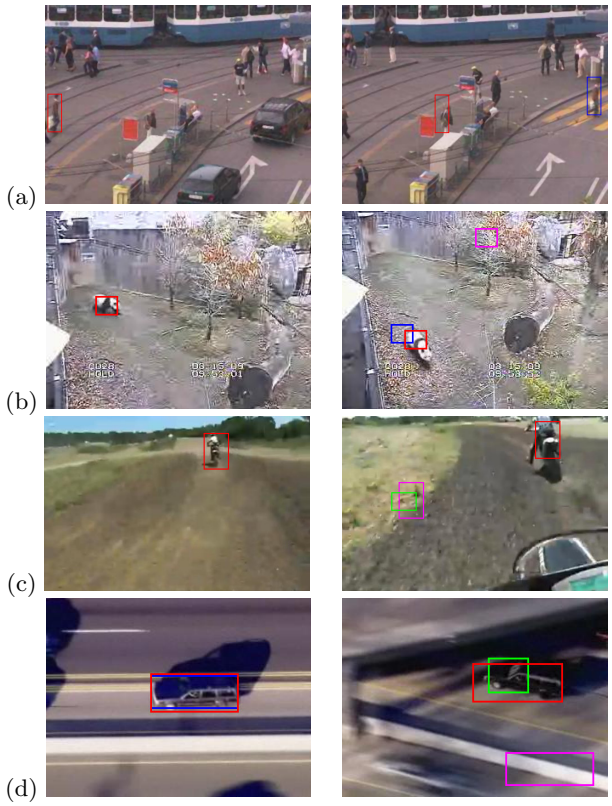


Fig. 5: Sequences used in the experiments: first and last frames with the different tracking results displayed with different colors: MS with colorspace switching in pink, CT in red, and CT/MS in blue /green.

Mean-Shift tracking (see section 3.2), the impact of the feature vector in Covariance tracking (explained in section 3.3) and the cooperation of MS and CT, as proposed in 3.4.

The four following sequences (the first and last frames are displayed on figure 5) show various situations that illustrate the contribution of the proposed method:

1) *Pedxing* (Fig.5(c), 189 frames) has been used in [22, 40]. The target is a pedestrian who is crossing the street, and passes behind a panel.

2) *Panda* (Fig.5(b), 940 frames)[13] comes from the Kalal's dataset [17]. It shows a panda walking in a park. In the first frames of the sequence, the camera zooms out while the panda is walking. The target has a non-rigid motion, and its appearance changes during the sequence. It turns on itself, which makes both its shape and color distribution change. In addition, the panda passes behind the tree, and the background changes.

3) *Motocross* [17] (Fig.5(c), 2665 frames) is particularly difficult. The camera is mounted on a motocross which follows the motocross to be tracked. The relative motion between the two vehicles is tricky, because of jumps and speed variations. That is particularly difficult for the Mean-Shift tracking which doen't tolerate large inter-frame motion.

4) *Carchase* [17] (Fig.5(d), 2999 frames) shows a car chased by an helicopter. The car drives fast on the highway and is frequently occluded when it passes in a tunnel. It also passes close to similar cars which can disturb the tracking.

Note that the images are not colorful, which makes tracking non-trivial. In each case, the target is selected manually in the first frame.

[13] This sequence is available here: http://info.ee.surrey.ac.uk/Personal/Z.Kalal/tld.html

5.1 MS: Colorspace switching

A colorspace switching method has been designed in 3.2 to overcome the limitations of MS. This method is evaluated here on the sequence *Panda*, since the background and the target appearance change several times. The color chart of figure 6 is used to display the results of the different colorspaces. Note that the proposed switching method is shown in fuschia (*switch*). Figure 7 shows the bounding boxes at different times of the sequence, and figure 8 displays the trajectories. At time $t = 130$, the panda passes behind the tree and MS with HSV, RGB, the L_1 norm or Yuv have lost the target. At time $t = 590$, before passing a second time behind the tree, MS with Ohta and Lab and our method still produce a correct tracking. Finally, at time $t = 610$ the only remaining method is the proposed switching procedure. Unfortunately, due to the severe occlusion, the panda is lost a few frames later by all the methods. In section 5.3, the combination with the covariance will handle this kind of problems.

Figure 8 shows that the best colorspaces for this sequence are Lab, $Ohta$ and XYZ. The switching procedure has automatically selected an adapted colorspace (HSV, then Lab and XYZ), three times during the sequence. It finally leads to better results compared to the most adapted colorspaces. Thus, even if the colorspace is well chosen at the beginning of the sequence, its validity can vanish during the time, when the context changes.

5.2 CT: selecting the features

Section 3.3 has proposed different color and texture feature vectors to compute the covariance matrix. In order to evaluate their impact in terms of separability, table 8 collects for each sequence the percentage of frames for which the covariance matching has correctly re-acquire the initial target, for the following features: RGB, luminance L, HSV, L_1, with the classical gradient (g_x, g_y); L_1 and L with LBP variance instead of gradient. Obviously, the use of three color components doesn't lead mandatorily to a better distinctiveness, as shown by the results of RGB and HSV (except in the Carchase sequence). In addition, the features vectors based on HSV

Table 8: Evaluation of the distinctiveness of different features vectors in the covariance matrix. The table collects the percentage of frames for which the matching has correctly re-acquire the target.

Sequence	RGB g_x, g_y	L g_x, g_y	L LBP	HSV g_x, g_y	L_1 g_x, g_y	L_1 LBP
Pedxing	51.2	64.8	**88.9**	59.4	30.1	41.2
Panda	88.8	96.4	**97.3**	61.5	35.1	70.7
Motocross	46.0	**70.5**	50.2	47.8	37.0	16.1
Carchase	59.6	73.6	37.7	**78.2**	40.5	35.7

are sensitive at low saturation, because the hue is ill-defined. L_1 theoretically offers invariance against illumination changes, but the distinctiveness is reduced, as shown by the poor results whatever the texture feature (gradient or LBP).

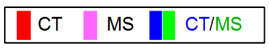
Finally, the combination of luminance with LBP provides the best results when the camera is still (sequences *Panda* and *Pedxing*). The use of luminance with gradients is also satisfactory in most situations. Besides, these two sets of features require a more compact covariance matrix (4×4 for L with LBP, and 5×5 with gradient) therefore the matching is faster, as seen by the benchmarks of section 4.

Considering these results, the following feature vector $[x, y, L, LBP]$ is selected for the next experiments.

5.3 Combination of trackers

This section evaluates the cooperation procedure between MS and CT. Mean-Shift is run with each colorspace and 32 quantization levels are used. The threshold $\mathcal{D}_{max}$ used to switch from one method to the other (see Fig.2), is fixed to 2 in all the sequences.

The tracking results are shown on figure 9 to figure 12. In each sequence, a different color is used for each tracking method:



The blue/green box displays the behavior of the cooperation method: blue when CT is run, and green when the MS is run. Let us analyze these results:

Pedxing (Fig.9). In this sequence, the CT (in red) fails at t=7574, while $CT + MS$ correctly tracks the pedestrian during the whole sequence (see t=7672). Indeed, when the target is not occluded, MS tracking is more precise than CT, and when the pedestrian passes behind the panel at t=7574, it is correctly handled by switching from MS to CT. Using CT alone, the tracking has been distracted by another pedestrian before the occlusion and finally has not been able to retrieve him.

Panda (Fig.10). At the beginning of the sequence, the Mean-Shift behaves correctly (from t=1 to 80), until the panda turns on itself and changes its appearance. At this moment, $\mathcal{D}_t$ increases and becomes higher than $\mathcal{D}_{max}$. It remains high during several successive frames (see t=190), when the back of the panda is viewed by the camera. The covariance tracking is run during all this period of time, while $\mathcal{D}_t$ remains high. When the panda turns on itself once again, its appearance is closer to its initial appearance, $\mathcal{D}_t$ decreases and control returns to Mean-Shift (t=300 and 400). The CT is run again when the panda turns on itself (frame 800), then MS is run when the panda walks straightforward (t=900). Finally, both CT and $CT + MS$ with colorspace switching have succeeded until the end of the sequence (frame 940). Figure 10 finally shows the trajectories of the three methods, when the sequence is played from the first frame to the last one (left images), and in the reverse order (right images).

Motocross (Fig.11). Due to the large motion of the target, the Mean-Shift fails in frame t=150 and never retrieves the target. None of the colorspaces allows to track the target correctly. In the cooperation procedure, the dissimilarity $\mathcal{D}_t$ allows to correctly detect the problem and selects the Covariance Tracking. When the target is retrieved, the Mean-Shift is able to track the target. By running CT alone, the tracking fails in some situations (see frame 220 for example).

Carchase (Fig.12). MS loses the target most of the time: when lighting conditions change (at t=365) or when the target is occluded (t=390) for example. The CT tracking can be distracted by another car, as shown in frames at t=50, 413, 1250, 2430, 2590. By making them cooperate, several severe difficulties are better tackled: total occlusion by a bridge between t=365 and 390), partial occlusion and blur at t= 413, scale change at t=2430. By associating MS and CT, the tracking is more precise than CT alone for *standard* situations (no occlusion), and more robust than MS when the target is lost due to occlusions or lighting changes.

Table 9 analyzes for each sequence, the number of cycles per second (cpp) spent by each section of the switching procedure and during the whole sequence: Mean-Shift and Covariance (which are never run simultaneously); the Model Update and the similarity criterion (performed in each frame). In addition, the table shows the percentage of frames for which MS and CT are run. Obviously, when the camera is still (sequences *Pedxing* and *Panda*), the MS is run most of the time, since the motion is regular. CT is run only to handle occlusions. When the camera is moving (sequences *Motocross* and *Carchase*), the motion of the target is tricky and more difficult to track. Therefore, CT is run more often.

The table shows the time spent by the tracking in ms per frame. For sake of comparison, in [40] the executing

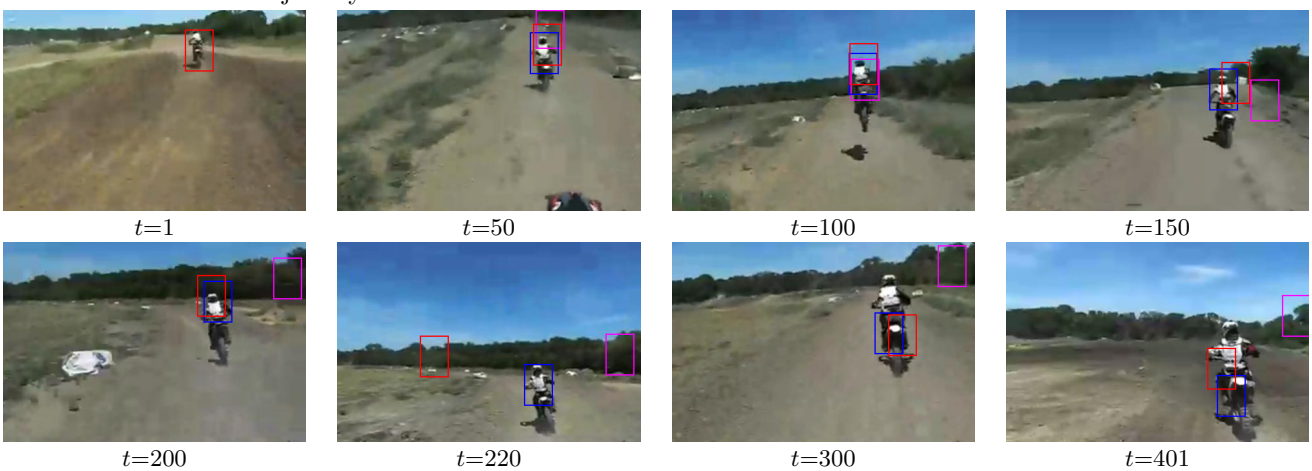
t=7484 t=7550 t=7574 t=7672

Fig. 9: Tracking results of sequence *Pedxing*

t=1 t=80 t=190 t=300

t=400 t=800 t=900 t=940

Fig. 10: Tracking results of sequence *Panda*, and trajectories. The left image shows the forward trajectory, the right one shows the reverse trajectory.



t=1 t=50 t=100 t=150

t=200 t=220 t=300 t=401

Fig. 11: Tracking results of sequence *Motocross*

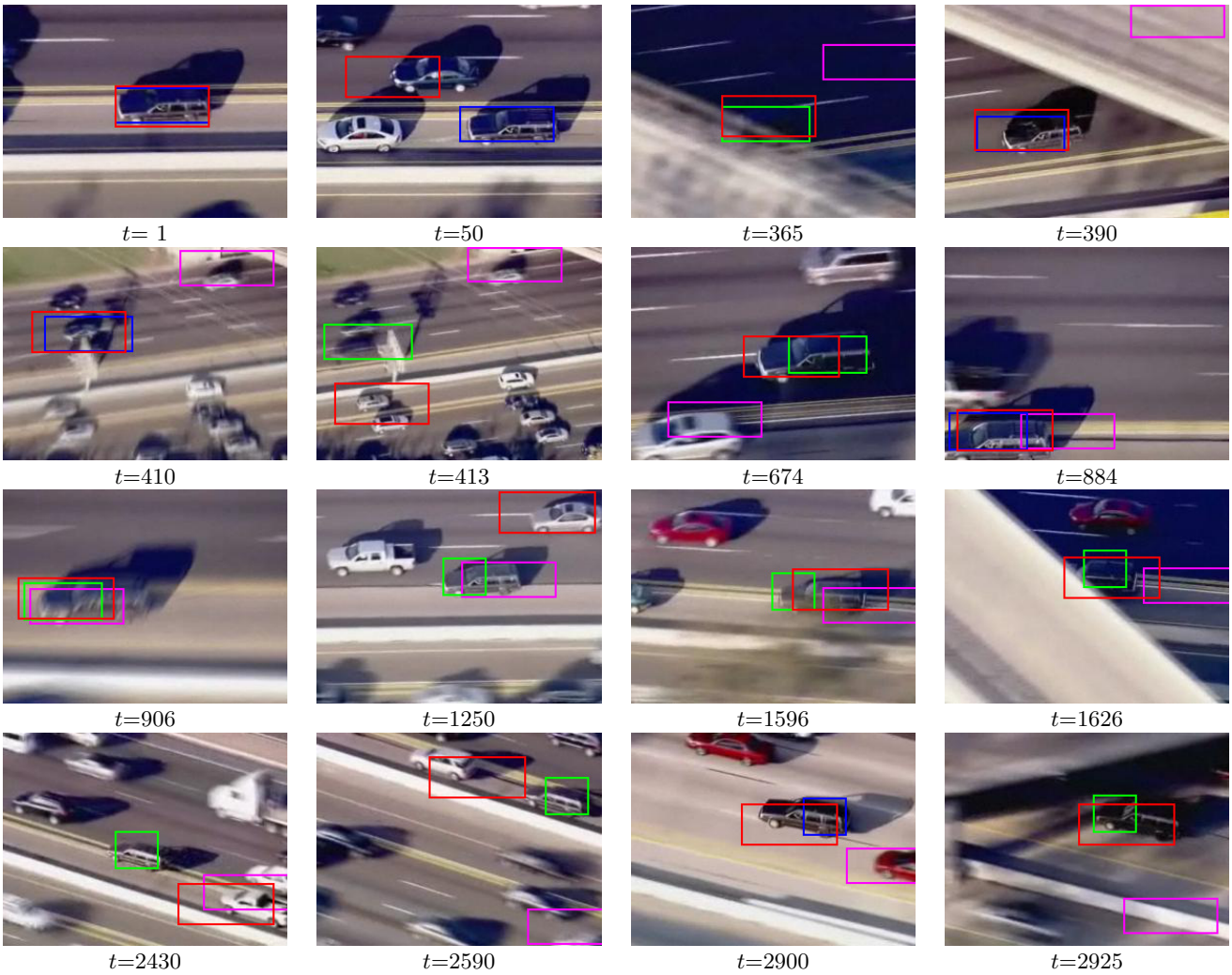


Fig. 12: Tracking results of sequence *Carchase*.

time goes from 20ms to 40ms in the *pedxing* sequence, and in [39], it goes from 31 to 56 ms depending on the sequence.

Finally, the color tracking by contextual switching has two advantages:

1. the tracking is robust because the Covariance Matching helps the Mean-Shift when it is deficient due to occlusion, tricky motion or appearance changes. In addition, the colorspace automatic selection guarantees a good behavior of the Mean-Shift whatever the colors of the target or the background;
2. the computation times are generally reduced compared to CT. Since MS is faster, the more often it is run, the faster the tracking. Consequently, the cooperation procedure is globally less time-consuming when the camera is still and the target is not occluded.

6 Conclusion

The paper has detailed two color tracking methods, Mean-Shift (MS) and Covariance Tracking (CT), that are known to be efficient for non-rigid targets. Two different strategies are jointly used to make the tracking adaptive to the context. First, the MS selects the colorspace which best separates the target with respect to the background. During tracking, the colorspace is changed only when needed, that is when the target is not distinct enough from its direct vicinity. Second, when MS fails, CT is run until the target is retrieved. Then, the control returns to MS. Without acceleration, the adaptiveness to the context would require a high computation time, and a trade-off should be made between robustness and speed.

The acceleration of both methods has been studied on different CPU, a special attention being granted to the most critical parts of the algorithms. Eight colorspaces have been tested and accelerated. The conversion times

Table 9: Analysis of the cooperation procedure. Results on Nehalem. The table shows for each sequence: the *Target Size* at initialization; the percentage of frames (*% frames*) for which the MS or the CT is run; the total number of cycles per pixel (*cpp*) spent by each algorithm section (MS, CT, Covariance model update, computation of the similarity) during the whole sequence.

Sequence	Target	Nb. of	Mean-Shift		Covariance		Model Update	Similarity	**Time**
	Size	frames	cycles ($\times 10^6$)	% fr	cycles ($\times 10^6$)	%fr	cycles ($\times 10^6$)	cycles ($\times 10^6$)	(ms/f)
Pedxing	93× 35	189	7.20	**69.97**	60.59	30.03	1.07	2.45	**10, 1**
Panda	23×28	940	2.48	**87.60**	8.74	12.40	0.57	0.33	**1,6**
Motocross	64×47	2665	5.07	11.87	30.04	**88.13**	0.005	1.06	**10,6**
Carchase	45×97	2999	0.99	13.33	19.19	**86.67**	1.03	1.44	**7,3**

become so low that it is possible to achieve all the color conversions simultaneously in real-time, and to choose the best color representation for tracking.

Of course, the execution times of MS and CT depend on the target size, therefore the acceleration is not predictable. However, the benchmarks have shown that both algorithms run in real-time on a quad-core, for very large object sizes.

In addition, these optimization results have shown that multi-threading is not obligatorily efficient. These conclusions will be useful since we plan to port our application on an embedded platform based on ARM Cortex A9 quad-core processor.

It is well known that MS is less efficient on gray images compared to color. The problem is not addressed in the paper, since CT performs well in this context, when the object is textured. In other respects, prediction will be investigated. It could help reducing the time required for searching the new location of the target in CT.

7 Acknowledgement

This research is supported by the European ITEA2 SPY project (Surveillance imProved sYstem). Further information is available on the website: http://www.itea2-spy.org/.

References

1. M. S. Allili and D. Ziou. Object tracking in videos using adaptive mixture models and active contours. *Neurocomputing for Vision Research: Advances in Blind Signal Processing*, 71:2001 – 2011, 2008.
2. R. V. Babu, P. Pérez, and P. Bouthemy. Robust tracking with motion estimation and local kernel-based color modeling. *IVC*, 25(8), 2007.
3. S. Bak, E. Corvee, F. Brémond, and M. Thonnat. Person re-identification using spatial covariance regions of human body parts. In *IEEE AVSS*, pages 435–440, 2010.
4. H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up Robust Features (SURF). *Computer Vision and Image Understanding: CVIU*, 110(3):346–359, June 2008.
5. S. Bouchafa and B. Zavidovique. c-velocity: A flow-cumulating uncalibrated approach for 3d plane detection. *Int. Jour. of Computer Vision*, 97(2):148–166, 2012.
6. J-Y. Bouguet. Pyramidal implementation of the lucas kanade feature tracker description of the algorithm. *In Practice*, 1(2):1–9, 2000.
7. S. Castillo, T. Judd, and D. Gutierrez. Using eye-tracking to assess different image retargeting methods. In *Proceedings of the ACM SIGGRAPH Symposium on Applied Perception in Graphics and Visualization*, APGV '11, pages 7–14, New York, NY, USA, 2011. ACM.
8. R.T. Collins, Y. Liu, and M. Leordeanu. Online selection of discriminative tracking features. *IEEE Trans. on PAMI*, 2005.
9. D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Trans. on PAMI*, 25:564–577, 2003.
10. A.I. Comport, E. Malis, and P. Rives. Accurate Quadrifocal Tracking for Robust 3D Visual Odometry. In *IEEE ICRA*, Rome, Italy, April 2007.
11. C.Yang, R.Duraiswami, and L.Davis. Efficient mean-shift tracking via a new similarity measure. In *IEEE Computer Society*, pages 176–183, 2005.
12. T. Gevers and A.W.M. Smeulders. Color-based object recognition. *Pattern Recognition*, 32(3):453–464, 1999.
13. M. Gouiffès, F. Laguzet, and L. Lacassagne. Color connectedness degree for mean shift tracking. In *ICPR*, Istanbul, 2010.
14. D-Y Huang, W-C Hu, and M-H Hsu. Adaptive skin color model switching for face tracking under varying illumination. In *ICICIC*, pages 326–329, 2009.
15. G. Iannizzotto. Competitive combination of multiple eye detection and tracking techniques. *IEEE Trans on Industrial Electronics*, 58(8):3151–3159, 2011.
16. I. Ishii, T. Ichida, and Q. Guand T. Takaki. 500-fps face tracking system. *Journal of Real-Time Image Processing*, pages 1–10, 2012.
17. Z Kalal, J Matas, and K Mikolajczyk. P-N Learning: Bootstrapping Binary Classifiers by Structural Constraints. *CVPR*, 2010.
18. C. Keller, C. Hermes, and D. Gavrila. Will the pedestrian cross? probabilistic path prediction based on learned motion features. *Pattern Recognition*, 6835:386–395, 2011.
19. Andreas Koschan and Mongi A. Abidi. *Digital Color Image Processing*. Wiley-Interscience, New York, NY, USA, 2008.
20. L. Lacassagne, A. Manzanera, and A. Dupret. Motion detection: Fast and robust algorithms for embedded systems. In *IEEE ICIP*, pages 3265 –3268, nov. 2009.
21. F. Laguzet, M. Gouiffès, L. Lacassagne, and D. Etiemble. Automatic color space switching for robust tracking. In *IEEE ICSIPA*, pages 295–300. IEEE, 2011.
22. Schindler K. & Gool L. V. Leibe, B. Coupled detection and trajectory estimation for multi-object tracking. In *Int. Conf. on Computer Vision*, pages 115–122, 2007.
23. I. Leichter. Mean shift trackers with cross-bin metrics. *IEEE Trans. on PAMI*, 34(4):695–706, April 2012.

24. Y. Liu, G. Li, and Z. Shi. Covariance tracking via geometric particle filtering. *EURASIP Journal on Advances in Signal Processing*, 2010:1–10, 2010.
25. D.G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.
26. C. Montes, C. Wong, J. Ziegert, and L. Mears. Vision-based tracking of a dynamic target with application to multi-axis position control. *Journal of Real-Time Image Processing*, pages 1–16, 2012.
27. T. Ojala, M. Pietikainen, and T. Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans. on PAMI*, 24(7):971–987, 2002.
28. E-J. Ong. Robust facial feature tracking using shape-constrained multiresolution-selected linear predictors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:1844–1859, September 2011.
29. F. Porikli, O.Tuzel, and P. Meer. Covariance tracking using model update based on lie algebra. In *IEEE Computer Vision and Pattern Recognition*, pages 728–735, 2006.
30. A. Romero, M. Gouiffès, and L. Lacassagne. Feature points tracking adaptive to saturation. In *IEEE ICSIPA*, pages 277–282, 2011.
31. J. Shi and C. Tomasi. Good features to track. Technical report, Cornell University, November 1993.
32. D. Song, B. Zhao, and L. Tang. Mean-shift algorithm fused with corner feature and color feature for target tracking. *Systems Engineering and Electronics*, 34(1):199–203, 2012.
33. S.Rastegar, M.Bandarabadi, Y.Toopchi, and S.Ghoreishi. Kernel based object tracking using metric distance transform and rvm classifier. In *AJBAS (3)'09*, pages 2778–2790, 2009.
34. H. Stern and B. Efros. Adaptive color space switching for face tracking in multi-colored lighting environments. In *5th IEEE International Conference on Automatic Face and Gesture Recognition*, FGR '02, page 249, 2002.
35. H. Stern and B. Efros. Adaptive color space switching for tracking under varying illumination, 2005.
36. O. Tuzel, F. Porikli, and P. Meer. Region covariance: A fast descriptor for detection and classification. In *ECCV 2006*, volume 3952, pages 589–600. Springer Berlin / Heidelberg, 2006.
37. A. Tyagi, J. W. Davis, and G. Potamianos. Steepest descent for efficient covariance trackin. In *WMVC*, 2008.
38. T. Vojíř and J. Matas. Robustifying the flock of trackers. In *16th Computer Vision Winter Workshop*, pages 91–97. Graz Univ. of Tech., 2011.
39. F. Wang, S. Yu, and J. Yang. Robust and efficient fragments-based tracking using mean-shift. *International Journal of Electronics and Communications*, 2009.
40. J. Wang and Y. Yagi. Switching local and covariance matching for efficient object tracking. In *19th International Conference on Pattern Recognition*, 2008.
41. Y. Wu, J. Cheng, J. Wang, H. Lu, J. Wang, H. Ling, E. Blasch, and L. Bai. Real-time probabilistic covariance tracking with efficient model update. *IEEE Trans. on Image Processing*, 21(5):2824 – 2837, 2012.
42. Y. Wu, B. Wu, J. Liu, and H. Q. Lu. Probabilistic tracking on riemannian manifolds. In *ICPR*, pages 1–4, 2008.