Pattern Recognition Letters 32 (2011) 411–422

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

Tree-structured image difference for fast histogram and distance between histograms computation

Séverine Dubuisson [*]

Laboratoire d'Informatique de Paris 6 (LIP6), Université Pierre et Marie Curie (UPMC), 4 place Jussieu, 75005 Paris, France

ARTICLE INFO

Article history:
Received 22 February 2010
Available online 5 November 2010
Communicated by R. Davies

Keywords:
Fast histogram computation
Integral histogram
Histogram-based distance

ABSTRACT

In this paper we present a new method for fast histogram computing and its extension to bin to bin histogram distance computing. The idea consists in using the information of spatial differences between images, or between regions of images (a current one and a reference one), and encoding it into a specific data structure: a tree. The histogram of the current image or of one of its regions is then computed by updating the histogram of the reference one using the temporal data stocked into the tree. With this approach, we never need to store any of the current histograms, except the reference image ones, as a preprocessing step. We compare our approach with the well-known Integral Histogram one, and obtain better results in terms of processing time while reducing the memory footprint. We show theoretically and with experimental results the superiority of our approach in many cases. We also extend our idea to the computation of the Bhattacharyya distance between two histograms, using a similar incremental approach that also avoid current histogram computations: we just need histograms of the reference image, and spatial differences between the reference and the current image to compute this distance using an updating process. Finally, we demonstrate the advantages of our approach on a real visual tracking application using a particle filter framework by improving its correction step computation time.

1. Introduction

Histograms are often used in image processing for feature representation (colors, edges, etc.). The complex nature of images implies a large amount of information to be stored in histograms, requiring more and more computation time. Many approaches in computer vision require multiple comparisons of histograms for rectangular patches of an input image. Each one is developed for a specific application, such as for image retrieval (Halawani and Burkhardt, 2005; Zhong and Defee, 2007), contrast enhancing (Caselles et al., 1999; Arici et al., 2009) or object recognition (Gevers, 2001; Laptev, 2010) (and many others). In such approaches, we dispose a reference histogram and try to find the region of the current image whose histogram is the most similar. The similarity is given by a measure that has to be computed between each target histogram and the reference one. This implies the computation of a lot of target histograms and similarity measures, that can be very time consuming, and may also need a lot of information to store. The main difficulty in such approaches is then to reduce the computation time, while using small data structures, requiring less memory.

In this article, we first propose a new fast histogram computation by using a data structure only coding the pixel differences between two frames of a video sequence. This data structure is then used to update the histograms of the first frame, computed during a preprocessing step. With such an approach, we never need to store any histogram (except those of the first frame) and our representation is compact because it only contains the information of differences between two frames. The main advantages of our approach are that it is not strongly dependent on the histogram quantization (i.e. number of bins), it is fast to compute (comparing to other approaches) and compact. We secondly propose an incremental version of the Bhattacharyya distance that uses the information of differences encoded in our data structure. We show this is possible to derive any Bhattacharyya distance between a precomputed histogram of a region in the first frame (the reference one) and the histogram of the corresponding region into the current frame, without needing to compute this current histogram. Section 2 reviews some of the previous works on fast histogram and similarity measure between histogram computations. Section 3 presents our method and its application to fast histogram computation. We extend our reasoning to a fast Bhattacharyya distance computation just using one histogram, and only updating this distance from a frame to another one. Section 4 gives some theoretical considerations about time computations and size of storage needed, and compare the proposed approach, for both contributions, with

* Fax: +33 0 1 44 27 74 95.
E-mail address: Severine.Dubuisson@lip6.fr

0167-8655/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2010.11.002

another one based on the Integral Histogram approach. In Section 5, experimental results show the benefit of our approach. In Section 6 we illustrate the capability of our method on a real application: object tracking using particle filtering. Finally, we give concluding remarks in Section 7.

2. Previous works

An image yields a distribution color space by mapping each pixel of the image to its color. Binning this probability distribution is a way to summarize it, and the applied quantization (bin size) controls how much the distribution is summarized: the resulting quantized structure is called histogram. Without any quantization, the histogram H of an image I is given by $H(k) = \sum_{x=1}^{N}\sum_{y=1}^{M}\{I(x,y) = k\}$, where $I(x,y)$ is a pixel, $k = 0,\ldots,L-1$ its value (in our case, this is a grayvalue, but we can also consider color histograms, gradient orientation histograms, etc.) and $N \times M$ the size of the image. If we now consider the histogram is divided into B bins $b = 1,\ldots,B$, then, a quantized histogram is defined by $H(b) = \sum_{x=1}^{N}\sum_{y=1}^{M}\{I(x,y) = k,\ k \in [(b-1)\frac{L}{B}, b\frac{L}{B}[\}$. An histogram H is then computed into a region R by browsing its pixels, yielding a global complexity of $\mathcal{O}(|R|)$, where $|R|$ is the number of pixels in the region. If lots of histograms have to be computed locally around a set of salient points, it can be advantageous to use the histogram of a nearby region and to update it to obtain the histogram of the current region, instead of computing all the histograms. This is particularly necessary in a lot of applications in image or video processing, such as image retrieval, spatial image filtering (contrast enhancing, denoising), or temporal image filtering (tracking, detection), when trying to find displacements of objects between frames. An important issue is here the comparison of images, that can be made by comparing their histogram, using a similarity measure or a distance. In practice, bin-to-bin distances, like the Euclidean, the χ^2 or the Bhattacharyya distances are considered as the simplest way to measure quickly the similarity between two histograms. The Bhattacharyya distance is believed to be the absolute similarity measure for frequency coded data, such as histograms, and is then a lot used for histogram comparisons, but if we need real-time treatments, this is necessary to accelerate its computation. This can be done by reducing the histogram computation time and/or their similarity measure computation, and a lot of works have been proposed, mainly for the first task.

One of the first work on the reduction of the redundancy in histogram computation was proposed in (Tang et al., 1979), in the context of image filtering (median filter). Considering the histogram H_R of a region R, the histogram of a region Q is computed by keeping the histogram of their intersection region, removing the pixels of R that do not belong to Q, and adding those from Q that do not belong to R. This approach can be very efficient if the two considered regions have a large intersection area. Recently this method has been improved by Perreault et al. (2007), also in the context of median filtering, similar in spirit to Tang et al. (1979), but using median filtering properties to optimize computations: they break up the histogram into the union of the columns of the image, and while filtering it, all histograms can be kept up to date in constant time with a two-step approach. In (Sizintsev et al., 2008) the authors propose the distributive histogram, that extends a part of the work presented in (Sizintsev et al., 2008) but generalizes it to many applications. They use the property for disjoint regions R and Q that is $H(R \bigcup Q) = H(R) + H(Q)$, and also maintain one histogram by column, but the difference is they update histograms not only using previously computed columns, like in other approaches, but also using rows. Their approach can also easily be adapted to non-rectangular regions and multi-scale processing that is not the case of previous approaches and then a strong advantage. One can also mention the approach (Hagyard et al., 1997), dedicated to morphological operations on images: they use a window similar to the structural element's one, that only considers the new pixels (i.e. not already computed ones) to recursively maintain the maximum or minimum in the window for then performing dilation or erosion operations. A fast way to compute histograms in terms of time computation is the Integral Histogram (Porikli, 2005) (IH), that is now used in many applications needing massive histogram computations by localized searches, especially in recent tracking algorithms (Adam et al., 2006; Wang et al., 2007). This approach, inspired from integral image (Viola and Jones, 2001), consists in computing the histogram of any region of an image only using four operations (two additions and two subtractions). IH is a cumulative function whose cells $\text{IH}(r_i,c_j)$ contain the histogram of the region of the image containing its r_i first rows and c_j first columns. Each cell is given by $\text{IH}(r_i,c_j) = I(r_i,c_j) + \text{IH}(r_i-1,c_j) + \text{IH}(r_i,c_j-1) - \text{IH}(r_i-1,c_j-1)$. Once IH has been computed over all cells, we also can derive any histogram of a sub-region only using four elementary operations, see Porikli (2005) for more details. For example, the histogram of a $w \times h$ region R with pixel (r_i,c_j) as bottom right corner is given by $H_R = \text{IH}(r_i,c_j) - \text{IH}(r_i-h,c_j) - \text{IH}(r_i,c_j-w) + \text{IH}(r_i-h,c_j-w)$. The main drawback of IH is the large amount of data that need to be stored. For an $N \times M$ image, the size of the array IH is $N \times M \times B$, where B is the number of bins of the histogram. All the methods previously exposed (we can find a good comparative study of some of them in (Sizintsev et al., 2008)) try to reduce the histogram computation time by avoiding making two times the same computation, and then exploiting redundancy properties. But they do not consider another important question: which one of the computations are really necessary?

A lot of works also concern the acceleration of the computation of the similarity between histograms, especially for tracking or detection applications. This is due to the fact that we need more and more information to achieve good tracking accuracy, that drastically increases the computation cost. Recent tracking applications concern for example articulated body (Koch et al., 2009; Nejhum et al., 2008), in which the object to track is decomposed into several regions, or also consider both the foreground and the background (Jeyakar et al., 2008; Hu et al., 2008), and for each one, lots of histograms and similarity measures are computed. Recent works also concern the combination of multiple visual cues (color, shape, motions, etc.) to achieve robust object detection or tracking (Badrinarayanan et al., 2007; Breitenstein et al., 2009; Leichter et al., 2009). If we know the experimental conditions, it is easy to construct histograms and similarity measures adapted to the task (Qiu and Han, 2008; Guha et al., 2002), and then to accelerate computation times. However, only few works directly concern the decreasing of the computation time of similarity measures between histograms, because most of them choose to reduce only the histogram computation time, and then to integrate them into their framework (Peihua, 2006; Adam et al., 2006). For all of these approaches, histograms are always computed, and as previously said, the main goal is to avoid redundant computations, not the undesirable ones. In a recent work (Yao et al., 2010) the authors add temporal information into Bhattacharyya distance computation to improve tracking the quality in of particle filter framework. In their case, they embed an incremental similarity matrix that handles target changes over time in the Bhattacharyya distance formulation. Their goal is not to accelerate the computation time, but to improve the robustness of the similarity measure.

In that sense, our approach is totally different than previous exposed ones: we never need to encode histograms (except the reference ones, as a preprocessing step), but only the temporal differences between two images, encoded into a specific data structure, that is then used to update histograms or distances between histograms. The size of this data structure, the histogram

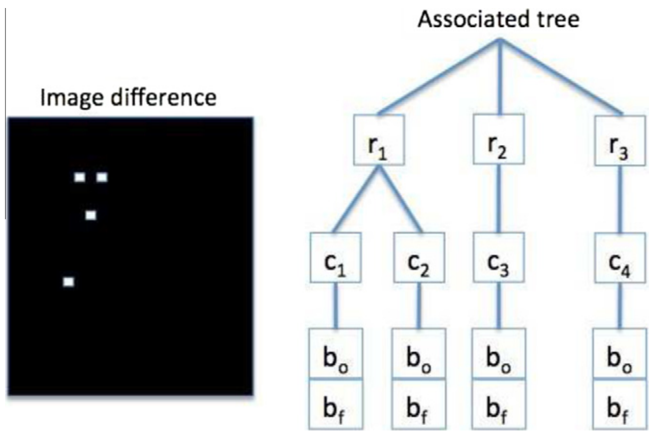

Fig. 1. Construction of the data structure associated with the image reference D, on the left. For each non-zero value pixel of D, we store in a tree its row number r_i, column number c_j and original and final bins, respectively b_o and b_f.

and Bhattacharyya distance computation times mainly depend on the variations between frames. This method is exposed in the next section.

3. Proposed approach: temporal structure

In this section, we describe our temporal structure, and the way we use it for fast histogram and Bhattacharyya distance computation. Assume that we have a reference Integral Histogram IH (from the reference image I_t), and want to compute any histogram in a new image $I_{t+\delta}$ by only using IH and temporal variations between I_t and $I_{t+\delta}$. Temporal variations are given by the image difference and we encode them using a tree data structure with height $h_T = 3$. Then, for each changing pixel (r_i, c_j), between I_t and $I_{t+\delta}$, located in the row r_i and the column c_j, we store r_i at the level $h = 1$ of the tree and c_j at the level $h = 2$. Leaf nodes contain, for each pixel (r_i, c_j), the difference between I_t and $I_{t+\delta}$, expressed by (i) the initial bin it was belonging to in H (extracted from I_t), and (ii) the bin it belongs to in $I_{t+\delta}$. Fig. 1 shows a basic example of the construction of this data structure. On the left, the image difference between I_t and $I_{t+\delta}$ shows only four different pixels, situated in three different rows r_1, r_2 and r_3, and four different columns c_1, c_2, c_3 and c_4. For each pixel (r_i, c_j), we also have to store its original and new bins, respectively b_o and b_f. Algorithm 1 summarizes the construction of our temporal structure T.

Algorithm 1. Temporal data structure construction

```
$T \leftarrow \{\}$
Compute the image difference $D = I_t - I_{t+\delta}$
for all $D(r_i, c_j) \neq 0$ do
  $b_o \leftarrow$ bin of $I_t(r_i, c_j)$; $b_f \leftarrow$ bin of $I_{t+\delta}(r_i, c_j)$
  if $b_o \neq b_f$ and node $r_i$ does not exist in $T$ then
    Create branch $r_i - c_j - (b_o b_f)$ in $T$
  else
    Add branch $c - (b_o b_f)$ to node $r$ in $T$
  end if
end for
```

3.1. Temporal Histogram

Once we have the reference histogram H and the difference tree T, we can derive any histogram of a region R in $I_{t+\delta}$, as described in Algorithm 2. We then just need to browse the data structure T to determine if some pixels have changed in this region between the two images (then, corresponding nodes have been created in T). For each changing pixel, we can derive the histogram of R from H by removing one from its bin b_o and adding one to its bin b_f. This is a very simple but efficient way to compute histograms because we just perform the necessary operations (where changes between frames have been detected).

Algorithm 2. Histogram computation of a region R

```
Extract the sub-tree $T_R$ from $T$, containing changing pixels in $R$
    between the two frames
for all node branch $r_i - c_j - b_o - b_f$ in $T_R$ do
  $H(b_o) \leftarrow H(b_o) - 1$
  $H(b_f) \leftarrow H(b_f) + 1$
end for
```

3.2. Temporal Bhattacharyya distance

If we consider two normalized histograms P and Q extracted from an image of size $N \times M$, with b_i (respectively q_i) the ith bins of P (respectively Q), $i = 1, \ldots, B$, the Bhattacharyya distance d^{bt} between P and Q is given by Bhattacharyya (1943):

$$d^{\text{bt}} = \sqrt{1 - \sum_{i=1}^{B} \sqrt{P(b_i)Q(q_i)}} \tag{1}$$

This can also be written as $d = (d^{\text{bt}})^2 - 1 = -\sum_{i=1}^{B} \sqrt{P(b_i)Q(q_i)}$.

As we said previously, when there is a bin difference between two pixels, we have to remove in H one from the original bin b_o and to add one to the new one b_f. If we consider normalized histograms, we have to remove $\frac{1}{NM}$ from b_o and to add $\frac{1}{NM}$ to b_f (this constant need to be precomputed only one time). Considering Eq. (1) we have to evaluate:

$$\sqrt{H(b_o)\left(H(b_o) - \frac{1}{NM}\right)} = \sqrt{H(b_o)}^2 - \left(\left(\sqrt{H(b_o)} - \sqrt{H(b_o) - \frac{1}{NM}}\right)\sqrt{H(b_o)}\right)$$
$$\sqrt{H(b_f)\left(H(b_f) + \frac{1}{NM}\right)} = \sqrt{H(b_f)}^2 - \left(\left(\sqrt{H(b_f)} - \sqrt{H(b_f) + \frac{1}{NM}}\right)\sqrt{H(b_f)}\right)$$

If we suppose we dispose the histogram of a region R of an image I_t, and that only one pixel changed its bin in R between I_t and $I_{t+\delta}$, the update $d_{t+\delta}$ can be written as:

$$\begin{aligned} d_{t+\delta} = &-\left(\sum_{i=1}^{B} \sqrt{H(b_i)}^2\right) \\ &+ \left(\sqrt{H(b_o)} - \sqrt{H(b_o) - \frac{1}{NM}}\right)\sqrt{H(b_o)} \\ &+ \left(\sqrt{H(b_f)} - \sqrt{H(b_f) + \frac{1}{NM}}\right)\sqrt{H(b_f)} \\ &= d_t + c_o + c_f \end{aligned} \tag{2}$$

where

$$c_o = \left(\sqrt{H(b_o)} - \sqrt{H(b_o) - \frac{1}{NM}}\right)\sqrt{H(b_0)}$$
$$c_f = \left(\sqrt{H(b_f)} - \sqrt{H(b_f) + \frac{1}{NM}}\right)\sqrt{H(b_f)}$$

are the update coefficients for one changing pixel (i.e. from bin b_o to bin b_f). The update of the Bhattacharyya distance is then given by:

$$(d^{\text{bt}}_{t+\delta})^2 = d_t + \delta + 1 = d_t + c_0 + c_f + 1 \tag{3}$$

Then, for each changing pixel, the Bhattacharyya distance can be updated using Eq. (3). The initial value of d^{bt} is fixed to 0 because there is no temporal changes in the reference frame, then

$d_0 = (d^{\mathrm{bt}})^2 - 1 = -1$. Note that this incremental version performs an exact computation of the Bhattacharyya distance between histograms. Algorithm 3 summarizes the idea, where $A = \frac{1}{NM}$ has been precomputed. A big advantage of this new way of writing the Bhattacharyya distance formulation is that square root values can be precomputed: $1 \leqslant b_o \leqslant B$ and $1 \leqslant b_f \leqslant B$ are bins, we can then store into a B-size array all the B possible $H(b)$ values ($b \in \{1,\ldots,B\}$), and then do not need to compute at each iteration these square root operations. This is possible but more time consuming with the classical formulation of Eq. (1), that necessitates to compute and store all possible $\sqrt{P(b_i)Q(q_i)}$.

Algorithm 3. Bhattacharyya distance update for a region R

```
Extract the sub-tree $T_R$ from $T$, containing changing pixels in $R$
    between the two frames $I_t$ and $I_{t+\delta}$
Precompute $\sqrt{H(b)}$ values, $b \in \{1,\ldots,B\}$
$d_t \leftarrow -1$
for all branch $r_i - c_j - b_o - b_f$ in $T_R$ do
    $c_o \leftarrow (\sqrt{H(b_o)} - \sqrt{H(b_o) - A})\sqrt{H(b_0)}$
    $c_f \leftarrow (\sqrt{H(b_f)} - \sqrt{H(b_f) + A})\sqrt{H(b_f)}$
    $d_t = d_t + c_0 + c_f$
    $(d^{\mathrm{bt}}_{t+\delta})^2 = d_t + 1$
end for
```

4. Theoretical study: memory and computation cost

Integral Histogram (IH) is, in our opinion, the best in the sense that it requires low computation times and is flexible enough to adapt to many applications, therefore it is the one we have chosen for comparison with our approach (reader can however find a good comparative study in (Sizintsev et al., 2008) between existing approaches and the Integral Histogram one).

In this section, we then compare our approach with IH in terms of number of operations necessary to compute histograms or distances between histograms, and size of storage needed for the required data structures. Here we consider the current image $I_{t+\delta}$ and the reference one I_t, of size $N \times M$, and B the number of bins in the histograms. The histogram of the reference image I_t has to be computed as a preliminary step for both approaches: we then do not consider this common step, neither the access to any of the sub-histograms of I_t that is also necessary for both approaches. We also do not consider the allocation operations for the two data structures (an array for Integral Histogram and a tree for Temporal Histogram), but this is clear that the tree needs less allocation operations than an array, for a fixed number of pixels, because it only stores four values per changing pixel, whereas IH stores a whole histogram per pixel. The determination of the bin of a current pixel requires one division and one floor: we call $f_b = \mathrm{div}(k,B)$ this operation (where k is a pixel value, and div the integer division). We also call a an addition (or a subtraction). The access to a cell of the array (for Integral Histogram) or of the tree (for Temporal Histogram) of the current image $I_{t+\delta}$ requires computing an offset, corresponding to a pointer addition (operation a). In our tests on Matlab®, we found $f_b \approx 8a$, by considering $B = 2^x$: we then will use this approximation to simplify our formulations. Both methods require two steps:

1. the data structure construction, then
2. the new histogram computation.

We first consider and compare independently both steps. Next, we call IH Integral Histogram and TH Temporal Histogram.

4.1. Construction of data structures

For IH, we need to browse each one of the NM pixels $I_{t+\delta}(r_i,c_j)$ of the image, determine its bin value (one operation f_b), and compute the histogram using four additions (four operations a for data access, then four operations a to add or subtract these values, see Section 2), for each of the B bins of the histogram. This part then needs a total number of operations of:

$$(n_d)_{\mathrm{IH}} = (8a + f_b)NMB \approx 16aNMB$$

This number of operations is constant.

For TH, we first need to find non-zero values in the image difference D (NM operations a). By scanning D in the lexicographic order, we then create a branch in the tree data structure for each non-zero value: let s be the total number of non-zero value pixels ($s \leqslant (N \times M)$). For each of the s changing pixels, we have to determine its new bin (one operation f_b) and stock the two bin values b_o and b_f into the tree (2 operations of addition, corresponding to pointer additions). The number of operations needed for the construction of the tree is then:

$$(n_d)_{\mathrm{TH}} = s(f_b + 2a) + NMa \approx a(10s + NM)$$

Thus, to compare with IH, we have to consider two special cases:

- In the best case, all the pixels in the image difference are zero-valued pixels: we need $(n_d)^{\mathrm{best}}_{\mathrm{TH}} = NMa$ operations to construct T.
- In the worst case, all the pixel values of the image difference are different from zero, the construction of T can be done using a total number of operations of:

$$(n_d)^{\mathrm{worst}}_{\mathrm{TH}} = NM(f_b + 2a) + NMa = NM(3a + f_b) \approx 11aNM$$

Even in the worst case (i.e. all the pixels have changed), the number of operations necessary for the construction of T is less than the one necessary for the Integral Histogram construction. The gain g using TH is significant even in this worst case, equal to $\frac{16}{11}B \approx \frac{3}{2}B$ operations, then increasing with B. In a general way, we have $\frac{3}{2}B \leqslant g \leqslant 16B$. It should also be noted that $(n_d)_{\mathrm{TH}}$ does not directly depend on the number B of bins of the histogram because we do not encode any histogram (and so do not need to browse all the bins), but only temporal changes between images. That is one of the advantages of our approach.

4.2. Histogram computations

4.2.1. One histogram computation

For both approaches, we consider the problem of computing the histogram of any region R of a new image $I_{t+\delta}$ knowing histograms in I_t.

For IH we just need two additions and two subtractions between values stored in the data structure (four operations a to access the data, then four to sum them), for each of the B bins of the histogram (see Section 2). Then, to compute any histogram of $I_{t+\delta}$, we need a constant number of operations:

$$(n_c)_{\mathrm{IH}} = 8aB$$

This is a bit more complicated for TH. We first need to extract the region R from T, if it does exist, i.e. if pixels have changed between I_t (we know its histograms, see the introduction of this section) and $I_{t+\delta}$. Then, for each of the s_R differences in R ($s_R \leqslant s$ if R is a subregion of $I_{t+\delta}$, otherwise $s_R = s$), we have to remove one from the bin b_o (two operations a to access b_o, then one operation a for the substraction) and add one to the bin b_f (two operations a to access b_f, then one operation a for the addition). The computation of a new histogram requires a total number of operations of:

$$(n_c)_{\mathrm{TH}} = 6as_R$$

We then need to consider the two following special cases:

- In the best case, there is no difference between the two considered regions: we need $(n_c)_{\mathrm{TH}}^{\mathrm{best}} = 0$ operation.
- In the worst case, all the pixel values have changed between the two regions, and an histogram computation requires a number of operations of:

  $$(n_c)_{\mathrm{TH}}^{\mathrm{worst}} = 6a|R|$$

  where $|R|$ is the number of pixels in R. Moreover, if $R = I_{t+\delta}$, then $(n_c)_{\mathrm{TH}}^{\mathrm{worst}} = 6aNM$.

The efficiency of our approach for the new histogram computation (without considering data structure construction times, see Section 4.2.2 for the total computation evaluation) then depends on the size of R and more precisely on the number of changing pixels between I_t and $I_{t+\delta}$ belonging to R. In the general case, we have:

$$(n_c)_{\mathrm{TH}} < (n_c)_{\mathrm{IH}} \quad \text{if } 6as_R < 8aB \Longleftrightarrow s_R < \frac{4}{3}B$$

We then conclude that TH is better as long as the number of changing pixels is less than $\frac{4}{3}$ times the number of bins of the computed histogram.

4.2.2. Total computation

The total histogram computation time in a region of the new image $I_{t+\delta}$ is fixed for IH:

$$(n_t)_{\mathrm{IH}} = (8a + f_b)NMB + 8aB \approx 8aB(2NM + 1)$$

For TH, it depends on two major factors: (i) the number s of changing pixels between I_t and $I_{t+\delta}$ and (ii) the number s_R of changing pixels between $(R)_{I_t}$ and $(R)_{I_{t+\delta}}$. We need a total number of operations:

$$(n_t)_{\mathrm{TH}} = NMa + s(f_b + 2a) + 6as_R \approx a(NM + 10s + 6s_R)$$

As previously, we may distinguish two cases:

- In he best case, there is no difference between the considered regions: T is empty and we need $(n_t)_{\mathrm{TH}}^{\mathrm{best}} = NMa$ operations, corresponding to the image difference computation.
- In the worst case, all the pixel are different between I_t and $I_{t+\delta}$ (so they are between $(R)_{I_t}$ and $(R)_{I_{t+\delta}}$), and we then need a total number of operations:

  $$(n_t)_{\mathrm{TH}}^{\mathrm{worst}} = NM(3a + f_b) + 6a|R| \approx a(11NM + 6|R|)$$

  If $R = I_{t+\delta}$, $(n_t)_{\mathrm{TH}}^{\mathrm{worst}} = NM(3a + f_b) + 6aNM = NM(9a + f_b) \approx 17aNM$.

IH depends on the size $N \times M$ of the image and on the number B of bins of the histogram. TH depends on the number of changing pixels s between I_t and $I_{t+\delta}$ but also on the size of the region on which we compute the histogram. But the data structure construction step requires less operations for TH (see Section 4.1).

Proposition 1. *The total time computing for one histogram computation is always less with Temporal Histogram approach, than with the Integral Histogram one.*

Proof. Let's compute $(n_t)_{\mathrm{TH}}^{\mathrm{worst}} - (n_t)_{\mathrm{IH}}$:

$$(n_t)_{\mathrm{TH}}^{\mathrm{worst}} - (n_t)_{\mathrm{IH}} \approx 17aNM - 8aB(2NM + 1) \approx \underbrace{\underbrace{17aNM - 16aBNM}_{\leqslant 0 \text{ if } B \geqslant 2} - 8aB}_{\leqslant 0 \text{ if } B \geqslant 2}$$

As the quantization of an histogram must be up to one, we deduce that $(n_t)_{\mathrm{TH}}^{\mathrm{worst}} < (n_t)_{\mathrm{IH}}$: Temporal Histogram approach gives the lower computation time for one histogram computation. □

4.3. Storage

We now compare the quantity of information necessary for both approaches.

For IH, we need one B-size array for each pixel (r_i, c_j), corresponding to the histogram of the region from rows 1 to N and columns 1 to M. Note that to avoid overflows, we can compute one Integral Histogram per bin. We then need a constant-size array, containing a total number of cells of:

$$(c)_{\mathrm{IH}} = NMB$$

For TH we use a tree T as data structure whose size depends on the number s of changing pixels between images I_t and $I_{t+\delta}$. If we call n_r the number of rows in $I_{t+\delta}$ containing changing pixels, the number of nodes of T is:

$$(c)_{\mathrm{TH}} = n_r + 3s$$

i.e. n_r for the rows, and 3 nodes for each changing pixel. Here again, we can distinguish two cases:

- In the best case, there is no difference between regions, T is empty: $(c)_{\mathrm{TH}}^{\mathrm{best}} = 0$.
- In the worst case, all the pixels are different, and the size if the required data structure T is:

  $$(c)_{\mathrm{TH}}^{\mathrm{worst}} = N + 3NM = N(1 + 3M)$$

Proposition 2. *Temporal Histogram necessitates less memory footprint than Integral Histogram does for any detection, comparison or recognition task based on histograms.*

Proof. We have $(c)_{\mathrm{TH}}^{\mathrm{worst}} - (c)_{\mathrm{IH}} = N(1 + 3M) - NMB$:

$$(c)_{\mathrm{TH}}^{\mathrm{worst}} \leqslant (c)_{\mathrm{IH}} \quad \text{if } N(1 + 3M) \leqslant NMB \Longleftrightarrow B \geqslant \frac{1 + 3M}{M} \Longleftrightarrow B \geqslant 3$$

For detection, comparison or recognition tasks based on histograms, we may use histograms that are not too much quantized. In the most common cases, we choose $B = 8$ or $B = 16$, values greater than 3, and then $(c)_{\mathrm{TH}}^{\mathrm{worst}} < (c)_{\mathrm{IH}}$. Globally our histogram computation needs then less storage. □

Note that if we consider $(c)_{\mathrm{TH}} = n_r + 3s$, we can say it is more than probable that the number of changing pixels between the two images is less than the total number of pixels. At most, if all these changing pixels are located on different rows (negative scenario), we have $n_r + 3s = s + 3s = 4s$, so:

$$(c)_{\mathrm{TH}} \leqslant (c)_{\mathrm{IH}} \quad \text{if } 4s \leqslant NMB \Longleftrightarrow s \leqslant \frac{NMB}{4}$$

The more B, N or M increases, the more this is useful to use our approach instead of Integral Histogram one.

4.4. Temporal Bhattacharyya distance computation

Usually, the computation of the Bhattacharyya distance corresponds to a scalar product between two B-size vectors (i.e. the histograms). If we call p a product of square roots (i.e. $p = \sqrt{x}\sqrt{y}$, with $x \in \mathbb{N}$ and $y \in \mathbb{N}$), the cost of the Bhattacharyya distance computation is then pB. To optimize and avoid computing a lot of square roots, this is better to precompute all the square roots and stock them into an array, so that the cell i of this array contains $\sqrt{i}$, $i = 1, \ldots, N \times M$ (tests have shown this takes 32 ms for $N = M = 1024$, and 8 ms for $N = M = 512$). In this case, the Bhattacharyya distance computation corresponds to B products, in our tests approximatively equivalent to B additions:

$$(t)^{\mathrm{bt}} \approx Ba$$

We are now comparing Bhattacharyya distance computation time given by our approach and by the classical one (using Integral Histogram). Note that with our approach, we never compute any histogram: we directly update the Bhattacharyya distance using the temporal tree T. We then do not need to store anything, except for the first frame (the reference one), that is also necessary in the other approach.

The computation of the Bhattacharyya distance between two images I_t and $I_{t+\delta}$ with (IH + Bhat.) approach necessitates the following steps:

1. Compute the whole Integral Histogram of $I_{t+\delta}$ (the one of I_t was already computed in a previous iteration of the algorithm).
2. Compute H_t and $H_{t+\delta}$ the histograms of respectively I_t and $I_{t+\delta}$.
3. Compute the Bhattacharyya distance between H_t and $H_{t+\delta}$ (operation $(t)^{\mathrm{bt}}$).

See Section 4.2.2 for the estimation of the computation times of the two first steps. The total computation time is then:

$$((n_t)_{\mathrm{IH}})^{\mathrm{bt}} \approx (8a + f_b)NMB + 8aB + aB \approx aB(16NM + 9)$$

For Temporal Bhattacharyya (TB), the idea is to update the distance value by browsing the temporal tree T. The computation of Bhattacharyya distance between two images I_t and $I_{t+\delta}$ necessitates the following steps:

1. Construct the temporal tree T only containing differences between I_t and $I_{t+\delta}$ (histograms was computed for I_t in a previous iteration of the algorithm).
2. Update the Bhattacharyya distance between h_t and $h_{t+\delta}$ using Eq. (3).

See Section 4.2.2 for the total computation time of the first step. The update of Bhattacharyya distance (Eq. (3)) necessitates to access data (four operations a to access each b_o and b_f in T), then four products and four additions ($\approx 8a$). The total computation time in the complete image $I_{t+\delta}$ is then:

$$((n_t)_{\mathrm{TH}})^{\mathrm{bt}} \approx s(f_b + 2a) + NMa + (4a + 8a)s_R \approx a(10s + NM + 12s_R)$$

We can distinguish two cases:

- In the best case, there is no difference between regions, T is empty ($s = s_R = 0$):

$$((n_t)^{\mathrm{best}}_{\mathrm{TH}})^{\mathrm{bt}} \approx NMa$$

- In the worst case, all the pixels are different ($s = NM$ and $s_R = |R|$):

$$((n_t)^{\mathrm{worst}}_{\mathrm{TH}})^{\mathrm{bt}} \approx a(11NM + 12|R|)$$

If $R = I_{t+\delta}$, $((n_t)^{\mathrm{worst}}_{\mathrm{TH}})^{\mathrm{bt}} \approx 23aNM$.

Proposition 3. *The computation time of the Bhattacharyya distance between two histograms is always less with Temporal Bhattacharyya approach, compared to the classical one.*

Proof. Let's compute $((n_t)^{\mathrm{worst}}_{\mathrm{TH}})^{\mathrm{bt}} - ((n_t)_{\mathrm{IH}})^{\mathrm{bt}}$ in the worst case for our approach (all the pixels have changed between frames, and we compute the whole histogram of $I_{t+\delta}$):

$$((n_t)^{\mathrm{worst}}_{\mathrm{TH}})^{\mathrm{bt}} - ((n_t)_{\mathrm{IH}})^{\mathrm{bt}} \approx \underbrace{\underbrace{23aNM - 16aBNM}_{\leqslant 0 \text{ if } B \geqslant 2} - 9aB}_{\leqslant 0 \text{ if } B \geqslant 2}$$

B is always up to one, we then have $((n_t)^{\mathrm{worst}}_{\mathrm{TH}})^{\mathrm{bt}} < ((n_t)_{\mathrm{IH}})^{\mathrm{bt}}$: Temporal Bhattacharyya gives the lower computation times. In that special case, $\frac{((n_t)_{\mathrm{IH}})^{\mathrm{bt}}}{((n_t)^{\mathrm{worst}}_{\mathrm{TH}})^{\mathrm{bt}}} \approx \frac{16aBNM+9aB}{23aNM} \approx \frac{16B}{23}$: our approach is $\frac{16B}{23}$ times faster than the classical approach. $\square$

As we have shown, this time mainly depends on the number s_R of changing pixels between the two considered regions of the frames. Two other parameters are also very important for bin to bin histogram distance computation: the whole size $N \times M$ of the image and the number B of bins of the considered histograms.

All of these theoretical considerations about the number of operations and storage needed for both approaches will be checked with a number of experimental results in the next section.

5. Experimental results

5.1. Temporal Histogram vs. Integral Histogram

In this section, we systematically compare Integral Histogram (IH) with the proposed Temporal Histogram (TH), since no method has been proved to be more interesting than IH in terms of both computation time and storage: it would then not be relevant to perform comparisons with other methods based on this criteria. All the tests have been performed with Matlab, on a MacBook Pro 2.53 GHz Intel Core Duo. In the next subsections, we call computation of an histogram the two-steps process needed for both approaches: data structure construction and histogram computation. In Section 4, we have highlighted some parameters that we directly involved in our tests, such as the number B of bins of the histograms, the size $N \times M$ of the images, the number s of changing pixels in the whole image, and the number s_R of changing pixels in the considered region for the histogram computation. We also test the number of histograms we can compute in a same frame while well-performing IH approach. All computation times reported in this section correspond to the mean value over 100 different tests. Finally, we compare storage footprints.

5.1.1. Video sequences

Tests on different complete video sequences have been performed. In this section we only present those made on sequences "Walking" (15 frames of size 275×320), "Tennis" (89 frames of size 240×342) and "Parking" (231 frames of size 576×768), see examples of frames in Fig. 2. In these tests, we are interested in the total computation times (along all the sequence) needed for the computation of the histograms of 10 randomly chosen regions of size 50×50 in each $I_{t+\delta}$ ($\delta \geqslant 1$, and at least 10% of pixels have changed between considered regions) i.e. all over the sequences, depending on the number of bins. We can see in Fig. 3 that the computation times obtained with our approach are lower for each one of these sequences. This is in part due to the fact that the computation of the array in each frame of the Integral Histogram takes a lot of time, even if the histogram computation time (just requiring four operations) is small. This also shows that our approach stays stable with respect to the increasing number of bins B, contrary to Integral Histogram one, whose computation time increases with B (drastically for $B = 256$). This is due to the fact that, contrary to IH, the number B of bins does not affect a lot the computation time (see Section 4.2.2) of TH if there are only few changing pixels between two frames. As there is no "best" number of bins, and different bin numbers can reveal different features of the data, it is difficult to determine an optimal number

Fig. 2. A frame from, from left to right: "Walking", "Tennis" and "Parking" sequences.

of bins, without making strong assumptions about the shape of the distributions. With our approach, it is not necessary to make such assumptions. Finally we can see our approach can achieve a mean of 4 times more frames per second than IH does.

5.1.2. Image and size variations

The performance of our approach principally depends on the number s of changing pixels. The larger s, the more consuming the method is. We then have tested the computation time as a function of s and compared results with those obtained by the IH method.

In the first test, on the "Walking" video sequence, I_1 (first frame) is used as reference image and we evaluated the histogram computation time for the histogram of the whole next image (I_2), in which 25% of the pixels vary from the first frame. Results are shown in Fig. 4, for different values of B. We can see that the computation time stays stable with the increasing of B with TH, when the one of IH drastically increase.

For the second test we have generated synthetic $N \times N$ images for different values of N and compared times for the computation of the histogram with $B = 16$ bins of this whole image (no pixel variation) for both approaches. Comparative results (in seconds) are reported in Table 1, depending on N. The increase of N does not influence a lot our approach, while drastically decreasing IH performance. As no pixel have changed between the two considered frames, the small time computation increasing for TH is just due to the pixel scanning of the new image, for the image of differences computing, that takes more time for a large image than a small one: this explains why the time computation for TH is equal to 0.19 ms for $N = 256$ and to 12 ms for $N = 2048$ (≈64 times more computing, exactly corresponding to the size of frame ration between these two cases).

The third test consists in considering a 1024×1024 synthetic image and simulating a number s of changing pixels, then computing the histogram with $B = 16$ bins of this new image. We have compared the computation times between both approaches depending on the percentage of changing pixels: results are reported in Table 2. The computation time with our approach stays always below IH's.

5.1.3. Number of histogram computations

The most time consuming part of IH is the construction of the array. However this array allows computing very quickly any histogram (or any set of histograms) only using four operations per histogram. In this subsection, we have launched a massive number of histogram computations and compared both approaches in terms of computation times. The idea is to simulate the computation of target histograms in a search window around a precise position, such as in spatial filtering or temporal filtering (particle filtering for example). Test have been made on the "Walking" sequence. We have chosen to present the results for different values of B. Results are shown in Fig. 5. Once again the performance depends on the quantization of the histograms. For a strong quantization ($B = 2$ or 4), IH and TH become equivalent for 5000

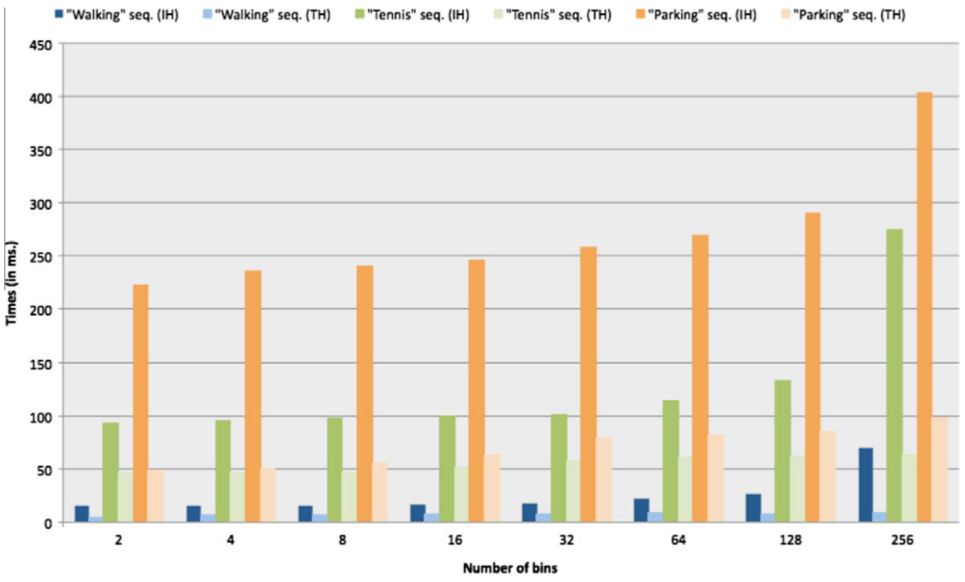

Fig. 3. Tests on different video sequences (see examples of frames in Fig. 2). Bar diagrams of computation times (in ms) of 10 randomly chosen histograms for both approaches all over the sequence (the number of frames depends on the sequence), for different sequences and increasing number of bins: "Walking" in blue, "Tennis" in green and "Parking" in orange (IH is represented as plain color, TH as transparency color). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

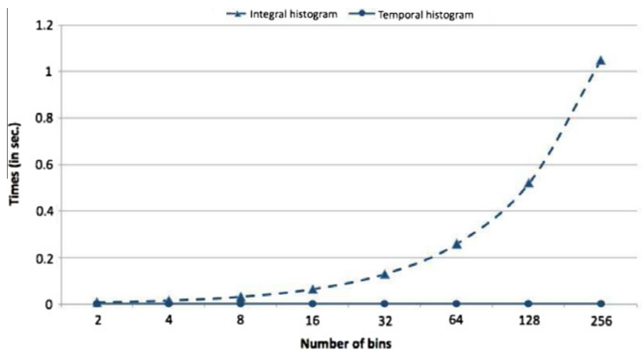

Fig. 4. Comparison of Integral Histogram (IH, dashed line) and Temporal Histogram (TH, plain line) time computation depending on quantization (value of B) of the histogram. Tests have been made on "Walking" sequence (275×320), between frames I_1 and I_2: 25% of the pixels have changed.

Table 1
Time computation (in seconds) of an histogram with $B = 16$ bins depending on the size of the image (no pixels have changed).

N	64	128	256	512	1024	2048
IH	0.002	0.009	0.03	0.14	0.58	2.34
TH	1.2×10^{-5}	4.7×10^{-5}	1.9×10^{-4}	7.64×10^{-4}	0.003	0.012

Table 2
Time computation (in sec.) of the histogram ($B = 16$) of a new 1024×1024 synthetic image depending on the percentage of changing pixels.

% Changing pixels	0%	5%	10%	25%	50%	75%	100%
IH	0.78	0.78	0.78	0.78	0.78	0.78	0.78
TH	0.003	0.0055	0.008	0.015	0.027	0.039	0.052

computations of histograms. For $B = 8, 16, 32$, methods are also equivalent for 5000 computations. For $B = 128$, 10,000 computations are needed, and 25,000 for $B = 256$. These results confirm theoretical results of Section 4, in which we have shown the more B increases, the more TH outperforms IH. This interesting result then shows that we can keep good results (compared to IH) with no need for a strong quantization of the histogram: TH does not need to approximate too much histograms to provide good computation times, which is a real advantage for histogram based search applications. We can however see one limitation of the proposed approach when dealing with too much histograms. It is clear that the performances of TH then depends on the number of changing pixels between the two considered regions on which we compute histograms, and that is the reason why our computation times increase with the number of computed histograms. However, for a small histogram quantization, we achieve very good results. Our TH is then suitable for visual tracking applications where it is better not to quantify histograms too much.

5.1.4. Storage

In this section we show that our approach does not need to store a lot of information, contrary to Integral Histogram. Table 3 reports the size (number of elements) required for the two data structures used for the test shown in Fig. 4 (between two consecutive frames of the sequence). The number of elements necessary for IH increases with the number of bins, according to the results of Section 4.3, in which we found $(c)_{\mathrm{IH}} = NMB$. For TH, it depends on the number of changing pixels between the two frames, $(c)_{\mathrm{TH}} = n_r + 3s$. A pixel is said to be changing between the two images if it changes its bins in the histogram. This notion then strongly depends on the histogram quantization: the more histogram is quantified, the less a pixel changes its bins between two images. That is the reason why the number of elements of our data structure indirectly depends on B. Note that the number of elements needed for IH does not change if images are really different, which is not the case for TH. We report in Table 4 the size of these data structures depending on the number s of changing pixels between two images generated as random 1024×1024 matrices (same experiments as in Section 5.1.2). We fixed $B = 16$. For this case, the number of elements necessary for Integral Histogram is fixed so that $(c)_{\mathrm{IH}} = NMB = 1024 \times 1024 \times 16 = 1.6 \times 10^7$. Even considering 10^6 changing pixels (i.e. 100% of the initial image) in the region where the histogram is computed, the number of elements needed to store it is always below the one IH needs.

In our opinion, TH is a good alternative to histogram computation in a lot of cases because it gives a compact description of temporal changes and lower histogram computation times. Moreover, we never need to store histograms (except for the reference image), that is a real advantage when working on video sequences (for these cases, the reference image is the first of the sequence, and the histogram computation can be seen as a preprocessing step).

5.2. Temporal Bhattacharyya distance vs. classical one

We compare computing time of the classical approach, consisting in extracting histograms using Integral Histogram approach, then computing the Bhattacharyya distance (we call (IH + Bhat.) this approach), and of our approach (TB).

5.2.1. Size of the image

In this test, we compare (IH + Bhat.) and TB computation times between two $N \times N$ images (randomly generated), depending on the number of changing pixels between frames (the number of bins is here fixed, $B = 16$). In the first line of Table 5, we have reported IH results (these results do not depend on the total number of changing pixels). In the other lines, we reported the results obtained with our approach, when, from top to bottom, 0%, 25%, 50% and 100% of the pixels have changed. The results emphasize the fact that our temporal data structure construction and use is very efficient because we do not compute distances when there is no changing pixels between frames: we just need to update the value of the distance (initialized to -1) if this is necessary. We can also see that even in the worst case (i.e. 100% of the pixels have changed between the two considered images), our time computations are lower. This is mainly due to the fact that the construction of the Integral Histogram takes a lot of time, although it is necessary to quickly compute any histogram in the image. In our case, we never compute any histogram, just update the distance value, that is a big gain of time.

As we can see in Table 5, it takes less computation time with our approach to compute Bhattacharyya distance between two histograms, even when 100% of the pixels changed their value. The mean gain over all values of N is 11.3. This confirms Proposition 3: our approach is approximatively $\frac{16B}{23} \approx 11.13$ times faster than (IH + Bhat.).

5.2.2. Number of bins

We now compare computation times of both approaches depending on the quantization of histograms (value of B). As it is well-known, bin-to-bin distances, such as the Bhattacharyya one, are not robust to the histogram quantization. Therefore, the number of bins is often limited to $B = 8$ to make a compromise between the discriminative power of the descriptors stored in the histogram and the robustness of the representation. Here again, we can see (Table 6) the advantages of our approach that is not drastically affected by the variation of this parameter, when the computation

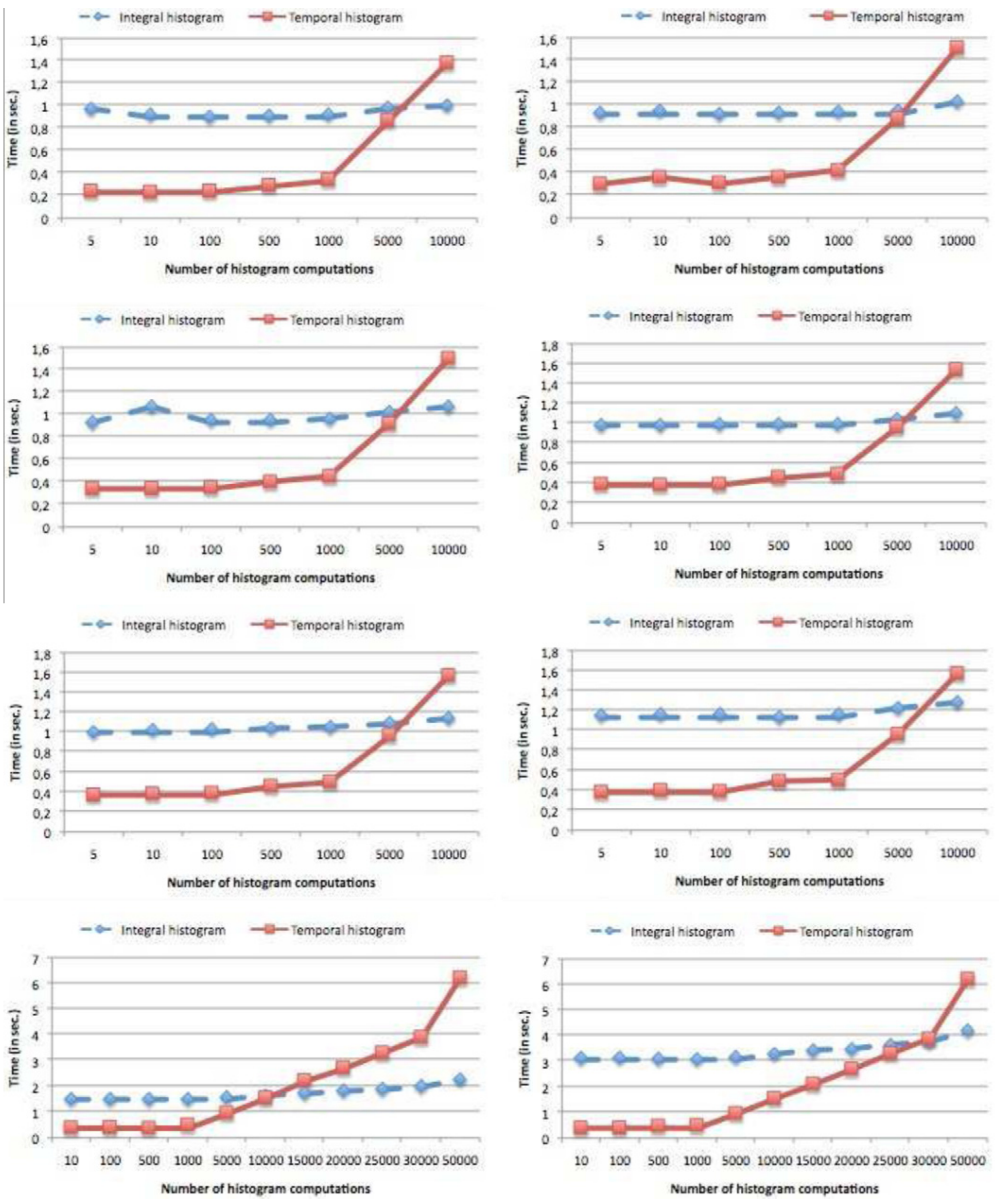

Fig. 5. Comparison of IH (dashed lines) and TH (plain lines) results for a massive number of histogram computations, for different values of B, from top to bottom, from left to right, 2, 4, 8, 16, 32, 64, 128 and 256.

Table 3
Size (number of elements) of data structures required for both approaches, depending on B, the test corresponds to the one of Fig. 4.

B	2	4	8	16	32	64	128	256
IH ($\times 10^5$)	1.76	3.52	7.04	14	28.1	56.3	112	225
TH ($\times 10^5$)	0.091	0.18	0.3	0.45	0.45	0.45	0. 453	0.453

Table 4
Size (number of elements) of data structures required for both approaches, depending on the number s of changing pixels between two images generated as random 1024×1024 matrices, for $B = 16$. Percentage (%) of changing pixels are reported in the second line of this table.

s	10^2	10^3	10^4	10^5	10^6
%	0.01	0.1	1	10	100
IH	1.6×10^7	1.6×10^7	1.6×10^7	1.6×10^7	1.6×10^7
TH	1.3×10^3	3.8×10^3	2.9×10^4	2.8×10^5	2.8×10^6

time of the other approach drastically increases with B. The construction of the Integral Histogram as well as Bhattacharyya distance computation, depending on B, play a major part in this increase.

Analyzing Tables 5 and 6 permits to establish that (IH + Bhat.) approach best performs for B and N minimal, and worst performs for N and B big. Let's compare these two special cases (100% of the pixels have changed between the two frames, that only does affect our approach results, and moreover is the worst case):

Table 5
Computation times of the Bhattacharyya distance between two histograms for both approaches (in ms) with $B = 16$ bins depending on the size of the region in which the histograms are computed and on the percentage of changing pixels s between these two regions (this only affects our approach).

N	16	32	64	128	256	512	1024	2048
(IH + Bhat.)	0.19	0.76	3.1	12.3	49	196.2	784.7	3138.7
TB (0%)	0.0007	0.0029	0.02	0.05	0.21	0.74	3.1	12.2
TB (25%)	0.0051	0.024	0.089	0.36	1.4	5.7	23	92
TB (50%)	0.009	0.035	0.14	0.57	2.3	9.2	36.8	147.1
TB (100%)	0.017	0.068	0.27	1.1	4.4	17.6	70.5	282

- $B = 2$, $N = 16$, (IH + Bhat.) approach takes 0.028 ms, when TB takes 0.017 ms.
- $B = 256$, $N = 2048$, (IH + Bhat.) approach takes 49.82 s, when TB takes 0.27 s.

These test have shown the advantages of our approach to accelerate distance between histograms computation times. Note that on color images, where B becomes larger (each color channel is quantized), Temporal Histogram is also very effective, especially compared to Integral Histogram, whose histogram construction becomes very slow. In the next section, we are considering a classical tracking problem, involving the computation of multiple distances between histograms in a same frame.

6. Integration into a particle filter: fast particle weight computation

A good tracker should be able to predict in which area of a new frame the object is. Among all the methods, one can cite probabilistic trackers. In such approaches, an object is characterized by a state sequence $\{x_k\}_{k=1,\ldots,n}$ whose evolution is specified by a dynamic equation $x_k = f_k(x_{k-1}, v_k)$. The goal of tracking is to estimate x_k given a set of observations. These observations $\{y_k\}_{k=1,\ldots,m}$, with $m < n$, are related to the states by $y_k = h_k(x_k, n_k)$. Usually, f_k and h_k are vector-valued, nonlinear and time-varying transition functions, and v_k and n_k are white Gaussian noise sequences, independent and identically distributed. Tracking methods based on particle filters (Gordon et al., 1993; Isard and Blake, 1998) can be applied under very weak hypotheses and consist of two main steps:

1. a prediction of the object state in the scene (using previous information), that consists in propagating particles according to a proposal function (see Chen (2003));
2. a correction of this prediction (using an available observation of the scene), that consists in weighting propagated particles according to a likelihood function.

Joint Probability Data Association Filter (JPDAF) (Vermaak et al., 2004) provides an optimal data solution in the Bayesian framework filter and uses a weighted sum of all measurements near the predicted state, each weight corresponding to the posterior probability for a measurement to come from an object. Between two observations, the set of particles evolves according to an underlying Markov chain, following a specific transition function. Given a new observation, each particle is assigned a weight proportional to its likelihood of belonging to a tracked object. New particles are randomly sampled to favor particles with higher likelihood. A classical approach consists in integrating the color distributions given by histograms into particle filtering (Pérez et al., 2002), by assigning a region (e.g. target region) around each particle and measuring the distance (e.g. the Bhattacharyya distance) between the distribution of pixels in this region and the one in the area surrounding the object detected in a previous frame (e.g. reference region). This context is ideal to test and compare our approach in a specific framework.

For this test we measure the total computation time of processing particle filtering in the first 60 frames of the "Rugby" sequence (240×320 frames, see a frame in Fig. 6): we are just interested in the $B = 16$ bin histogram computation time around each particle location, then the particle's weight computation using the Bhattacharyya distance, that is the point of our paper. In the first frame of the sequence, the validation region (of fixed size 50×70 pixels) containing the object to track (one rugby player) is manually detected. JPADF is then used along the sequence to automatically track the object using N_p particles. Then, the total computation times needed for each method is detailed below:

- For (IH + Bhat.): one Integral Histogram H_i in each frame $i = 1, \ldots, t$ of the sequence, then N_p target histograms (one for each particle) are computed using four operations on H_i, and, for each one, its Bhattacharyya distance to the reference histogram is evaluated.
- For TB: one Integral Histogram H (only in the first frame), one tree T_i construction in each frame $i = 1, \ldots, t$ of the sequence, then, the update of the computed Bhattacharyya distances for each of the N_p regions surrounding the particles.

Computation times are reported in Table 7 for different numbers N_p of particles used. Computation times are lower with our approach until $N_p = 5000$ (tests have shown that for $N_p = 8500$, computation times are the same for both methods). Note that, in practice, we do not need so much particles in a classical problem. Our approach permits real-time particle filter based tracking for a reasonable number of particles, which is a real advantage. Note that the purpose of this test was not to deal with tracking performances, we then do not give any results about tracking quality: we just want to show that integrating TH instead of IH into particle filter correction step can accelerate the process. Moreover, we have shown that for similar computation times, we can use more particles into the framework integrating TH. As it is well-known (Gordon et al., 1993) that the particle filter converges with a high number of particles N_p, we can argue that integrating TH into a particle algorithm improves visual tracking quality. Note that we have

Table 6
Computation times (in ms) of the Bhattacharyya distance between two histograms depending on the number B of bins of the histograms: 50% of the pixels have changed between the two considered 512×512 size images.

B	2	4	8	16	32	64	128	256
(IH + Bhat.)	24.5	49	98.1	196.2	392.3	784.7	1569.3	3138.7
TB	9.2	9.21	9.19	9.2	9.2	9.21	9.2	9.18

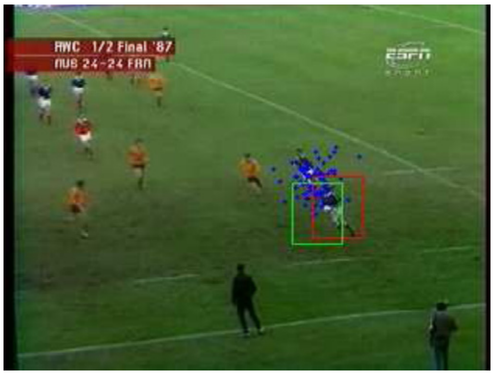

Fig. 6. Example frame of the "Rugby" sequence: the red rectangle is the validation region, the green one the target region associated to one particle. Blue crosses symbolize particle positions in the frame around which we compute histograms. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 7
Total computation times (in ms) and gain (in %) between both approaches for all histograms ($B = 16$) in the particle filter framework, depending on the number N_p of particles.

N_p	50	100	1000	5000	10,000
IH	47.2	47.4	47.5	50.48	53.59
TH	12.5	13.6	28.31	40.4	76.24
Gain	+73.5%	+71.3%	+40%	+19.9%	−42.3%

Table 8
Total computation times (in ms) and gain (in %) between both approaches for all Bhattacharyya distance computations in the particle filter framework, depending on the number N_p of particles.

N_p	50	100	500	1000	5000
IH	55	55.4	55.7	60.48	73.59
TH	14	19.6	28.31	45.4	96.24
Gain	+74.5%	+64.6%	+49.1%	24.9%	−30%

obtained the same kinds of results on different video sequences (people tracking on "Parking" sequence and ball tracking on "Tennis" sequence). Table 8 shows the total computation times (in ms) for all Bhattacharyya distance computations in the particle filter framework in one frame, depending on the number N_p of particles ($B = 16$), for both approaches. Here again, our approach gives better results. As the update of the Bhattacharyya distance requires more operations, the time computing for this test quickly increases with our approach, but keeps good results comparing with the other approach until $N_p = 2500$.

7. Conclusion

We have presented in this paper a new method for fast histogram computation, called Temporal Histogram (TH) and its extension to fast Bhattacharyya distance computation. The principle consists in never encoding histograms, but rather temporal changes between frames, in order to update a first preprocessed histogram. This technique presents two main advantages: we do not need a large amount of information to store whole histograms and it is less time consuming for histogram or distance between histograms computation. We have shown by theoretical and experimental results that our approach outperforms the well-known Integral Histogram in terms of total computation time and quantity of information to store. Moreover, the introduction of TH into the particle filtering framework has shown its usefulness for real-time applications in most common cases. Integral Histogram requires the computation of the accumulator array in each new image which takes a lot of time (never taken into account in the evaluation of computation times of the classical approaches). TH computes histogram only if necessary (i.e. some changes between images have been detected). We also have extended this reasoning to fast Bhattacharyya distance computation by defining a new incremental definition of this distance that does not necessitate the computation of histograms. Future works will concern the study of other distances (or similarity measures) between histograms to determine if we can use temporal information (i.e. differences) to update them. We also work on an incremental spatio-temporal definition of the Bhattacharyya distance that could be employed for spatial filtering.

References

Adam, A., Rivlin, E., Shimshoni, I., 2006. Robust fragments-based tracking using the integral histogram. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition, pp. 798–805.

Arici, T., Dikbas, S., Altunbasak, A., 2009. A histogram modification framework and its application for image contrast enhancement. IEEE Image Process. 18 (9), 1921–1935.

Badrinarayanan, V., Pérez, P., Clerc, F.L., Oisel, L., 2007. Probabilistic color and adaptive multi-feature tracking with dynamically switched priority between cues. In: IEEE Internat. Conf. on Computer Vision, Rio de Janeiro, Brazil, pp. 1–8.

Bhattacharyya, A., 1943. On a measure of divergence between two statistical populations defined by probability distributions. Bull. Calcutta Math. Soc. 35, 99–110.

Breitenstein, D., Reichlin, F., Leibe, B., Koller-Meier, E., Gool, L.V., 2009. Robust tracking-by-detection using a detector confidence particle filter. In: IEEE Internat. Conf. on Computer Vision, Kyoto, Japan.

Caselles, V., Lisani, J., Morel, J., Sapiro, G., 1999. Shape preserving local histogram modification. IEEE Trans. Image Process. 8 (2), 220–230.

Chen, Z., 2003. Bayesian filtering: from kalman filters to particle filters, and beyond. Tech. Rep., McMaster University.

Gevers, T., 2001. Robust histogram construction from color invariants. IEEE Trans. Pattern Anal. Machine Intell. 26, 113–118.

Gordon, N., Salmond, D.J., Smith, A., 1993. Novel approach to nonlinear/non-gaussian bayesian state estimation. Radar Signal Process., IEE Proc. F 140 (2), 107–113.

Guha, S., Koudas, N., Srivastava, D., 2002. Fast algorithms for hierarchical range histogram construction. In: Proc. 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, New York, USA.

Hagyard, D., Razaz, M., Atkin, P., 1997. Fast histogram method for fast greyscale morphology operations. In: IEEE Workshop on Nonlinear Signal and Image Processing.

Halawani, A., Burkhardt, H., 2005. On using histograms of local invariant features for image retrieval. In: IAPR Conf. on Machine Vision Applications, pp. 538–541.

Hu, J.-S., Juana, C.-W., Wanga, J.-J., 2008. A spatial-color mean-shift object tracking algorithm with scale and orientation estimation. Pattern Recognition Lett. 29 (16), 2165–2173.

Isard, M., Blake, A., 1998. Condensation-conditional density propagation for visual tracking. Internat. J. Comput. Vision 29, 5–28.

Jeyakar, J., Babu, R., Ramakrishnan, K., 2008. Robust object tracking with background-weighted local kernels. Comput. Vision Image Understanding 112 (3), 296–309.

Koch, C., Mauthner, T., Tilp, M., N, N.S., 2009. Evaluation of visual position estimation in beach volleyball. Internat. J. Perform. Anal. Sport 9 (13), 332–343.

Laptev, I., 2010. Improving object detection with boosted histograms. Comput. Vision Image Understanding 114, 400–408.

Leichter, I., Lindenbaum, M., Rivlin, E., 2009. Mean shift tracking with multiple reference color histograms. Image Vision Comput. 27 (5), 535–544.

Nejhum, S.S., Ho, J., Yang, M.-H., 2008. Visual tracking with histograms and articulating blocks. In: IEEE Internat. Conf. on Pattern Recognition, Anchorage, Alaska, USA.

Peihua, L., 2006. A clustering-based color model and integral images for fast object tracking. Signal Process.: Image Commun. 21 (8), 676–687.

Pérez, P., Hue, C., Vermaak, J., Gangnet, M., 2002. Color-based probabilistic tracking. In: ECCV '02: Proc. 7th European Conf. on Computer Vision – Part I. Springer-Verlag, London, UK, pp. 661–675.

Perreault, S., Hebert, P., 2007. Median filtering in constant time. IEEE Trans. Image Process. 16 (9), 2389–2394.

Porikli, F., 2005. Integral histogram: a fast way to extract histograms in cartesian spaces. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition, pp. 829–836.

Qiu, Q., Han, J., 2008. A new fast histogram matching algorithm for laser scan data. In: IEEE Conf. on Robotics, Automation and Mechatronics, Chengdu, China.

Sizintsev, M., Derpanis, K., Hogue, A., 2008. Histogram-based search: a comparative study. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition, pp. 1–8.

Tang, G., Yang, G., Huang, T., 1979. A fast two-dimensional median filtering algorithm. IEEE Trans. Acoust. Speech Signal Process., 13–18.

Vermaak, J., Godsill, S.J., Pérez, P., 2004. Monte carlo filtering for multi-target tracking and data association. IEEE Trans. Aerosp. Electron. Systems 41, 309–332.

Viola, P., Jones, M., 2001. Robust real-time object detection. Internat. J. Comput. Vision.

Wang, H., Suter, D., Schindler, K., Shen, C., 2007. Adaptive object tracking based on an effective appearance filter. IEEE Trans. Pattern Anal. Machine Intell. 29 (9), 1661–1667.

Yao, A., Wang, G., Lin, X., Chai, X., 2010. An incremental Bhattacharyya dissimilarity measure for particle filtering. Pattern Recognition 43, 1244–1256.

Zhong, D., Defee, I., 2007. Performance of similarity measures based on histograms of local image feature vectors. Pattern Recognition Lett. 28 (15), 2003–2010.