The computation of the Bhattacharyya distance between histograms without histograms

Conference Paper · August 2010

DOI: 10.1109/IPTA.2010.5586745 · Source: IEEE Xplore

1 author:

Séverine Dubuisson
Sorbonne Université
103 PUBLICATIONS **933** CITATIONS
SEE PROFILE

The computation of the Bhattacharyya distance between histograms without histograms

Séverine Dubuisson
Laboratoire d'Informatique de Paris 6, Université Pierre et Marie Curie
e-mail: Severine.Dubuisson@lip6.fr

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 and a reference one), and encoding it into a specific data structure: a tree. The Bhattacharyya distance between two histograms is then computed using an incremental approach that avoid histogram: 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. 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. 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.**

Keywords— **Histogram, Bhattacharyya distance**

I. 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 (image retrieval [16] or object recognition [7], *etc.*). 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. A lot of works 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 [9], both the foreground and the background [6], or try to combine multiple visual cues [2], [8] to achieve robust object detection or tracking, and for each one, lots of histograms and similarity measures are computed. If we know the experimental conditions, it is easy to construct histograms and similarity measures adapted to the task [13], [5], and then 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 [10], [1]. For all of these approaches, histograms are always computed, and the main goal is to avoid redundant computations, not the undesirable ones. In a recent work [15] the authors add temporal information into Bhattacharrya distance computation to improve tracking quality in the particle filter framework. In their case, they embed an incremental similarity matrix that handles target changes over time in the Bhattacharrya distance formulation to improve the robustness of the similarity measure, not to accelerate the computation time.

In this article, we first propose a new way of computing the Bhattacharyya distance between two histograms by using a data structure that only codes the pixel differences between two frames of a video sequence. With such an approach, we never need to store or compute any histogram and our representation 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. The size of this data structure, and Bhattacharrya distance computation times only depends on the variations between frames. This method is exposed in the next section. Section II presents our method and its application to fast Bhattacharyya distance computation only updating this distance from a frame to another one. Section III gives some theoretical considerations about time computations and size of storage needed, and compare the proposed approach with another one based on the Integral Histogram approach. In Section IV, some experimental results show the benefit of our approach. In Section V we illustrate the capability of our method on a real application: object tracking using particle filtering. Finally, we give concluding remarks in Section VI.

II. Proposed approach: temporal structure

In this section, we describe our temporal structure, and the way we use it for fast Bhattacharyya distance computations. Assume we dispose a reference image I_t and a current one $I_{t+\delta}$. Temporal variations are obtained by computing 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 stock 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 in I_t, and (ii) the bin it belongs to in $I_{t+\delta}$. Figure 1.(a) 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**
```

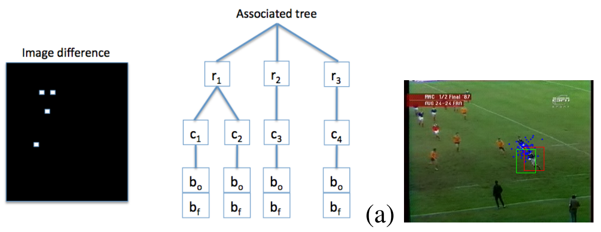

Fig. 1. (a) Construction of the data structure associated with the image reference D, on the left. For each non-zero value pixel of D, we store in a tree its row number r_i, column number c_j and original and final bins, respectively b_o and b_f. (b) Example frame of the "Rugby" sequence: the red rectangle 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.

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 i^{th} bins of P (respectively Q), $i = 1, \ldots, B$, the Bhattacharrya distance d^{bt} between P and Q is given by [3]:

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

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

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

where $c_o = \left(\sqrt{H(b_o)} - \sqrt{H(b_o) - \frac{1}{NM}}\right)\sqrt{H(b_0)}$ and $c_f = \left(\sqrt{H(b_f)} - \sqrt{H(b_f) + \frac{1}{NM}}\right)\sqrt{H(b_f)}$ are the update coefficients for one changing pixel (i.e. from bin b_o to bin b_f). The update of the the Bhattacharrya distance is then given by:

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

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). Then, for each changing pixel, the Bhattacharrya distance can be updated using Equation 3 (the initial value of d^{bt} is fixed to 0 because there is temporal changes in the reference frame, then $d_0 = (d^{\text{bt}})^2 - 1 = -1$). This is a very simple but efficient way to compute histograms because we just perform the necessary operations (where changes have been detected). Algorithm 2 summarizes the idea, where $a = \frac{1}{NM}$ as been precomputed.

Algorithm 2 Bhattacharrya 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 $I_{t+\delta}$
$d_t \leftarrow -1$
**for all** node branch $r_i - c_j - b_o - b_f$ in $T_R$ **do**
    $c_o \leftarrow \left(\sqrt{H(b_o)} - \sqrt{H(b_o) - a}\right)\sqrt{H(b_0)}$
    $c_f \leftarrow \left(\sqrt{H(b_f)} - \sqrt{H(b_f) + a}\right)\sqrt{H(b_f)}$
    $\left(d^{\text{bt}}_{t+\delta}\right)^2 = d_t + 1 + c_0 + c_f$
**end for**
```

III. Memory and Computation Cost

Integral histogram (IH) [12] is, in our opinion, the best in the sense that it requires low computation time 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 [14] between existing approaches and the Integral Histogram one). This approach consists in computing the histogram of any region of an image using only four operations (two additions and two subtractions). IH is a cumulative function whose cells $IH(r_i, c_j)$ 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 $IH(r_i, c_j) = I(r_i, c_j) + IH(r_i - 1, c_j) + IH(r_i, c_j - 1) - IH(r_i - 1, c_j - 1)$. Once IH has been computed over all cells, we can derive any histogram of a sub-region only using four elementary operations [12]. For example, the histogram of a $w \times h$ region R with pixel (r_i, c_j) as bottom right corner is given by $H_R = IH(r_i, c_j) - IH(r_i - h, c_j) - IH(r_i, c_j - w) + IH(r_i - h, c_j - w)$. 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. We do not consider the allocation operations for the two data structures (an array for Integral Histogram and a tree for Temporal Histogram), but 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 $f_b = \operatorname{div}(k, B)$ this operation (where k is a pixel value, and div is 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.

A. 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), for each of the B bins of the histogram. This part then needs a total and constant number of operations $(n_d)_{\text{IH}} = (8a + f_b)NMB \approx 16aNMB$.

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 s the total number of non-zero value pixels ($s \leq (N \times M)$). For each of the s changing pixels, we have to determine the old and new bins b_o and b_f (two operations f_b) and stock them into the tree (2 operations of addition, corresponding to pointer additions). The number of operations needed for the construction of the tree is then $(n_d)_{\text{TH}} = s(2f_b + 2a) + NMa \approx a(18s + NM)$. We can consider two special cases:

- In the best case, all the pixels in the image difference are zero-valued pixels: we need $(n_d)_{\text{TH}}^{\text{best}} = NMa$ 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 $(n_d)_{\text{TH}}^{\text{worst}} = NM(2f_b + 2a) + NMa = NM(3a + 2f_b) \approx 19aNM$.

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 $\frac{16}{19}B \approx \frac{4}{5}B$ operations, then increasing with B. In a general way, we have $\frac{4}{5}B \leq g \leq$ 16B. It should also be noticed that $(n_d)_{\text{TH}}$ does not 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), only temporal changes between images. That is one of the advantages of our approach.

B. 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 $(c)_{\text{IH}} = NMB$.

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 $(c)_{\text{TH}} = n_r + 3s$, 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: $(c)_{\text{TH}}^{\text{best}} = 0$.
- In the worst case, all the pixels are different, and the size if the required data structure T is $(c)_{\text{TH}}^{\text{worst}} = N + 3NM = N(1 + 3M)$.

Proposition 1: Temporal Histogram necessitates less memory footprint than Integral Histogram does for any detection, comparison or recognition task based on histograms.

Proof: Let's compute $(c)_{\text{TH}}^{\text{worst}} - (c)_{\text{IH}}$:

$$N(1 + 3M) - NMB \leq 0 \quad \text{if} \quad B \geq \frac{1 + 3M}{M} \Leftrightarrow B \geq 3$$

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 $(c)_{\text{TH}}^{\text{worst}} < (c)_{\text{IH}}$. Globally our histogram computation needs then less storage. ■

Note that if we consider $(c)_{\text{TH}} = n_r + 3s$, 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 $(c)_{\text{TH}} \leq (c)_{\text{IH}}$ if $4s \leq NMB \Leftrightarrow s \leq \frac{NMB}{4}$: the more B, N or M increases, the more this is useful to use our approach instead of Integral Histogram one.

C. Temporal Bhattacharrya distance computation

Usually, the computation of the Bhattacharrya distance corresponds to a scalar product between two B-size vectors (i.e. the histograms). If wee 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 Bhattacharrya 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 $\sqrt{i}$, $i = 1, \dots, N \times M$. In this case, the Bhattacharrya distance computation corresponds to B products, in out tests approximatively equivalent to B additions $(t)^{\text{bt}} \approx Ba$. We are now comparing Bhattacharrya distance time computation given by our approach and the classical one (using Integral Histogram). Note that with our approach, we never compute any histogram: we directly update the Bhattacharrya distance using the temporal tree T. We then do not need to stock anything, except for the first frame (the reference one), that is also necessary in the other approach.

The computation of the Bhattacharrya 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), this take $\approx 16aNMB$ operations (see Section III-A).
2) Compute H_t and $H_{t+\delta}$ the histograms of respectively I_t and $I_{t+\delta}$. 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. Then, to compute the histograms of I_t and $I_{t+\delta}$, it takes a constant number of operations $2 \times 8aB$.
3) Compute the Bhattacharrya distance between H_t and $H_{t+\delta}$ (operation $(t)^{\text{bt}}$).

The total computation time is then $((n_t)_{\text{IH}})^{\text{bt}} \approx (8a + f_b)NMB + 16aB + aB \approx aB(16NM + 17)$.

For Temporal Bhattacharrya (TB), the idea is to update the distance value by browsing the temporal tree T. The computation of Bhattacharrya 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 $I_{t+\delta}$, this takes $\approx a(10s + NM + 12s_R)$ operations (see Section III-A).
2) Update the Bhattacharrya distance between h_t and $h_{t+\delta}$ using Equation 3.

The update of Bhattacharrya (Equation 3) necessitates the access to 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 $((n_t)_{\text{TH}})^{\text{bt}} \approx s(2f_b + 2a) + NMa + (4a + 8a)s_R \approx a(18s + NM + 12s_R)$. We can distinguish two cases:

- In the best case, there is no difference between regions, T is empty ($s = s_R = 0$): $((n_t)_{\text{TH}}^{\text{best}})^{\text{bt}} \approx NMa$.
- In the worst case, all the pixels are different ($s = NM$ and $s_R = |R|$): $((n_t)_{\text{TH}}^{\text{worst}})^{\text{bt}} \approx a(19NM + 12|R|)$. If $R = I_{t+\delta}$, $((n_t)_{\text{TH}}^{\text{worst}})^{\text{bt}} \approx 31aNM$.

Proposition 2: The computation time of the Bhattacharrya distance between two histograms is always less with Temporal Bhattacharrya approach, compared to the classical one.

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

$$((n_t)_{\text{TH}}^{\text{worst}})^{\text{bt}} - ((n_t)_{\text{IH}})^{\text{bt}} \approx \underbrace{31aNM - 16aBNM - 17aB}_{\leq 0 \quad \text{if} \quad B \geq 2}$$

B is always bigger than one, we then have $((n_t)_{\text{TH}}^{\text{worst}})^{\text{bt}} < ((n_t)_{\text{IH}})^{\text{bt}}$: Temporal Bhattacharrya gives the lower computation times. In that special case, $\frac{((n_t)_{\text{IH}})^{\text{bt}}}{((n_t)_{\text{TH}}^{\text{worst}})^{\text{bt}}} \approx \frac{16aBNM + 9aB}{31aNM} \approx \frac{16B}{31}$: our approach is $\frac{16B}{31}$ times faster that the classical approach. ■

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 verified with a number of experimental results in the next section.

IV. Experimental results

A. Storage

Table 1 reports the size (number of elements) required for the two data structures used (between two consecutive frames of size 275×320). The number of elements necessary for IH increases with the number of bins, according to the results of Section III-B, in which we found $(c)_{\text{IH}} = NMB$. For TH, it depends on the number of changing pixels between the two frames, $(c)_{\text{TH}} = n_r + 3s$. 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 2 the size of these data structures depending on the percentage of changing pixels between two images generated as random 1024×1024 matrices. We fix $B = 16$. For this case, the number of elements necessary for integral histogram is fixed such that $(c)_{\text{IH}} = NMB = 1024 \times 1024 \times 16 = 1.6 \times 10^7$. Even considering 100% of changing pixels in the region where the histogram is computed, the number of elements needed to store it is always below the one IH needs. To our opinion, TH is a good alternative to histogram computation in a lot of cases

Table 1. Size (number of elements$\times 10^5$) of data structures required for both approaches, depending on B.

B	2	4	8	16	32	64	128	256
IH	1.76	3.52	7	14	28.1	56.3	112	225
T	0.09	0.18	0.3	0.45	0.45	0.45	0.45	0.45

Table 2. Size (number of elements) of data structures required for both approaches, depending on the percentage of changing pixels between two images generated as random 1024×1024 matrices, for $B = 16$.

%	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
T	1.3×10^3	3.8×10^3	2.9×10^4	2.8×10^5	2.8×10^6

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).

B. Temporal Bhattacharrya distance vs. classical one

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

1) Size of the region: In this test, we compare (IH+Bhat) and TB computation time 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 3, we 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 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 less. 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 3, it takes less computation time with our approach to compute Bhattacharrya distance between two histograms, even when 100% of the pixels changed their value. The meaning gain over all values of N is 8.27. This confirms Proposition 2: our approach is approximatively $\frac{16B}{31} \approx 8.25$ times faster than (IH+Bhat.).

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

Table 3. Computation times of the Bhattacharrya distance between two histograms of $N \times N$ size regions for both approaches (in ms) with $B = 16$ bins depending on the percentage of changing pixels s between them (this only affects our approach).

N	32	64	128	256	512	1024	2048
(IH+Bhat.)	0.78	3.2	12.6	50	202	808	3231
TB (0%)	0.003	0.01	0.05	0.2	0.74	3.2	12.6
TB (25%)	0.026	0.1	0.4	1.7	6.7	26	107
TB (50%)	0.049	0.2	0.78	3.2	12.6	50.5	202
TB (100%)	0.095	0.38	1.5	6.1	24.5	97.8	391.3

the Bhattacharrya 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 stocked in the histogram and the robustness of the representation. Here again, we can see (Table 4) the advantages of our approach that is not affected by the variations of this parameter, when the other approach's computation time drastically increases with B. The construction of the integral histogram as much as Bhattacharrya distance computation, depending on B, play a major part in this time increasing.

Table 4. Computation times of the Bhattacharrya 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 image size.

B	2	4	8	16	32	64	128	256
(IH+Bhat.)	25.2	50	101	202	404	808	1616	3231
TB	12.6	12.6	12.6	12.6	12.6	12.6	12.6	12.6

Analyzing Table 3 and Table 4 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 is the worst case): (1) if $B = 2$, $N = 16$, then (IH+Bhat.) approach takes 0.024 ms, when TB takes 0.012 ms, (2) if $B = 256$, $N = 2048$, then (IH+Bhat.) approach takes 51.69 seconds, when TB takes 0.2 second. These test have shown the advantages of our approach for fast distance between histograms computation. In the next section, we are considering a classical tracking problem, involving the computation of multiple distances between histograms in a same frame.

V. Integration into particle filter: fast particle weight computation

The goal of tracking is to estimate a state sequence $\{x_k\}_{k=1,\dots,n}$ whose evolution is specified by a dynamic equation $x_k = f_k(x_{k-1}, v_k)$ given a set of observations. These observations $\{y_k\}_{k=1,\dots,m}$, 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 [4] can be applied under very weak hypotheses and consist of two main steps: (i) a prediction of the object states in the scene (using previous information), that consists in propagating particles according to a proposal function, followed by (ii) a correction of this prediction (using an available observation of the scene), that consists in weighting propagated particles according to a likelihood function. 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 [11], by assigning a region (e.g. target region) around each particle and measuring the 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 Figure 1.(b)): we are just interested on the $B = 16$ bin histogram computation time around each particle locations, 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. For (IH+Bhat.), we need one integral histogram H_i in each frame $i = 1, \dots, t$ 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, we need one tree T_i construction in each frame $i = 1, \dots, t$ of the sequence, then, the update of the computed Bhattacharyya distances for each of the N_p regions surrounding the particles. Table 5 shows the total computation times (in ms) for all Bhattacharyya distances computations in the particle filter framework in one frame, depending on the number N_p of particles ($B = 16$), for both approaches. As the update of the Bhattacharyya distance requires more operations, the time computing for this test increases quickly for our approach, but keep good results comparing to the other approach until $N_p = 3500$. Our approach permits real-time particle filter based tracking for a reasonable number of particles, which is a real advantage. Moreover, we have shown than for similar computation times, we can use more particles into the frameworks integrating T. As it is well-known [4] that the particle filter converges with a high number of particles N_p, we can argue that integrating T into a particle algorithm improves visual tracking quality. Note that we have obtained same kinds of results on different video sequences.

Table 5. Total computation times (in ms) and time gain (in %) between both approaches for all Bhattacharyya distances 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
TB	14	19.6	28.31	45.4	96.24
Gain	+74.5%	+64.6%	+49.1%	+24.9%	-30%

VI. Conclusions

We have presented in this paper a new method for fast Bhattacharyya distance computation, called Temporal Bhattacharyya: we defined a new incremental definition of this distance that does not necessitates the computation of histograms before. The principle consists in never encoding histograms, but rather temporal changes between frames, in order to update a value. 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 our approach into the particle filtering framework has shown its usefulness for real-time applications in most common cases. Future works will concern the study of distances (or similarity measures) between histogram 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

[1] A. Adam, E. Rivlin, and I. Shimshoni. Robust fragments-based tracking using the integral histogram. In *Proc. IEEE CVPR*, pages 798–805, 2006.

[2] V. Badrinarayanan, P. Pérez, F. Le Clerc, and L. Oisel. Probabilistic color and adaptive multi-feature tracking with dynamically switched priority between cues. In *Proc. IEEE ICCV*, pages 1–8, Rio de Janeiro, Brazil, October 14-20 2007.

[3] A. Bhattacharyya. On a measure of divergence between two statistical populations defined by probability distributions. *Bulletin of the Calcutta Mathematical Society*, 35:99–110, 1943.

[4] Z. Chen. Bayesian filtering: From kalman filters to particle filters, and beyond. Technical report, McMaster University, 2003.

[5] S. Guha, N. Koudas, and D. Srivastava. Fast algorithms for hierarchical range histogram construction. In *Proc. ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, New-York, USA, 2002.

[6] J. Jeyakar, R.V. Babu, and K.R. Ramakrishnan. Robust object tracking with background-weighted local kernels. *Computer Vision and Image Understanding*, 112(3):296–309, 2008.

[7] I. Laptev. Improving object detection with boosted histograms. *Computer Vision and Image Understanding*, 114:400–408, 2010.

[8] I. Leichter, M. Lindenbaum, and E. Rivlin. Mean shift tracking with multiple reference color histograms. *Image and Vision Computing*, 27(5):535–544, 2009.

[9] S.M. Shahed Nejhum, J. Ho, and M.-H. Yang. Visual tracking with histograms and articulating blocks. In *Proc. IEEE ICPR*, Anchorage, Alaska, USA, June 24-26 2008.

[10] L. Peihua. A clustering-based color model and integral images for fast object tracking. *Signal Processing: Image communication*, 21(8):676–687, 2006.

[11] P. Pérez, C. Hue, J. Vermaak, and M. Gangnet. Color-based probabilistic tracking. In *Proc. IEEE ECCV*, pages 661–675, London, UK, 2002. Springer-Verlag.

[12] F. Porikli. Integral histogram: A fast way to extract histograms in cartesian spaces. In *Proc. IEEE CVPR*, pages 829–836, 2005.

[13] Q. Qiu and J. Han. A new fast histogram matching algorithm for laser scan data. In *Proc. IEEE CRAM*, Chengdu, China, September 21-24 2008.

[14] M. Sizintsev, K.G. Derpanis, and A. Hogue. Histogram-based search: A comparative study. *Proc. IEEE CVPR*, pages 1–8, 2008.

[15] A. Yao, G. Wang, X. Lin, and X. Chai. An incremental bhattacharyya dissimilarity measure for particle filtering. *Pattern Recognition*, 43:1244–1256, 2010.

[16] D. Zhong and I. Defee. Performance of similarity measures based on histograms of local image feature vectors. *Pattern Recognition Letters*, 28(15):2003–2010, 2007.