FAST MULTIPLE HISTOGRAM COMPUTATION USING KRUSKAL'S ALGORITHM

Raoul Berger, Séverine Dubuisson and Christophe Gonzales

Université Pierre et Marie Curie, Laboratoire d'Informatique de Paris 6
4 place Jussieu, Paris 75252 Cedex 05, France

ABSTRACT

In this paper, we propose a novel approach to speed-up the computation of the histograms of multiple overlapping non rotating regions of a single image. The idea is to exploit the overlaps between regions to minimize the number of redundant computations. More precisely, once the histogram of a region has been computed, this one can be used to compute part of the histogram of another overlapping region. For this purpose, an optimal computation order of the regions needs to be determined and we show how this can be obtained as the solution of a minimum spanning tree of a graph modeling the overlaps between regions. This tree is computed using Kruskal's algorithm and parsing it in a depth-first search manner determines precisely how the histogram of a region can be computed efficiently from that of its parent in the tree. We show that, in practical situations, this approach can outperform the well-known integral histogram both in terms of computation times and in terms of memory consumption.

Index Terms— Histogram, particle filter, spanning tree

1. INTRODUCTION

The complex nature of images implies that a large amount of data needs to be stored in histograms (colors, edges, *etc.*), thus requiring increasing computation times. Many approaches in computer vision require multiple comparisons of the histograms of rectangular patches of an input image. This applies in particular to Visual Tracking and the methodological framework of Particle Filtering [1], whose goal is to estimate the current state of a stochastic process given a set of past and present noisy observations using both recursive Bayesian filtering and Monte Carlo sampling. Monte Carlo simulation is used to represent and approximate the posterior density by a weighted sample of possible state realizations called particles. A classical approach to compute the particle weights consists in integrating the color distributions given by histograms into particle filtering [2]. This is done by assigning a region (the target region) around each particle and by measuring the distance between the distribution of pixels in this region and that in the area surrounding the object detected in a previous frame (the reference region). More often than not, particles' target regions are overlapping, which induces many redundant computations (i.e., some parts of histograms are computed several times). Fig. 1(a) illustrates this problem: a tracked person is modeled by a set of particles (blue plain squares), each one having its own target region (blue square). As can be seen, there exist some overlaps (in yellow) and the histograms of those regions should definitely be computed only once.

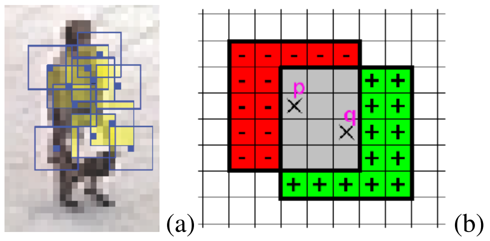

Fig. 1. (a) Overlapping problem: some particle's target regions have a nonempty intersection (yellow parts). (b) A case of two overlapping regions centered on locations p and q.

Many approaches have been proposed to speed-up histogram computations. A first method was proposed in [3], in the context of image filtering (median filter). The histogram of a region R^1 overlapping another one R^2 (whose histogram has already been computed) is computed by removing from the histogram of R^2 the pixels that do not belong to the intersection between R^1 and R^2 and by adding those that belong to R^1 but not to R^2. This approach can be very efficient if the two regions considered have a large intersection. Similar in spirit, the method proposed in [4] breaks up region R^1 into the union of its columns in the image, and all their histograms can be kept up to date in constant time with a two-step approach. In [5], the authors propose the distributive histogram based on a distributive property of disjoint regions combined with a per-column histogram maintenance and a row-based update of these column histograms. This approach can be easily extended to cope with non-rectangular regions and multi-scale processing. Another fast method designed specifically to compute large amounts of histograms is the Integral Histogram [6] (IH). This method is now used in many applications needing massive histogram computations, especially in recent tracking algorithms [7]. This approach, inspired from integral image, consists in first preprocessing the whole image storing some histogram in every image cell and, then, in computing the histogram of any region using only three arithmetic operations over these cell histograms (two subtractions and one addition) without resorting anymore to image parsing. Recently, temporal histogram has been proposed [8], that uses the information of spatial differences between regions of images. The current histogram is computed using previous ones and temporal changes between frames.

Except IH, the above approaches aim at minimizing the number of operations necessary to compute the histograms of a given pair of regions. The approach proposed in this paper is different in spirit: it considers the whole set of regions whose histograms are needed and determines the order in which the regions should be processed to minimize the number of operations to compute *all* the histograms, not only two of them. For this purpose, we model sets of overlapping regions by a graph whose nodes correspond to their centers and, using Kruskal's algorithm, we compute one of its minimum spanning tree which, when traversed in a depth-first manner, provides the order in which the histograms should be computed to minimize redundancies. The organization of the paper is as follows. Section 2 explains the principle of our approach. Section 3 provides experimental results, and concluding remarks and perspectives are finally given in Section 4.

2. PROPOSED APPROACH

Let V denote the set of N 2D points $\mathbf{x}^i = (x^i, y^i)$, which are the centers of the regions the histograms of which we look for. Our approach, described in Algo. 1, is divided into three main steps, that are detailed in the next subsections:

1. Construct the Delaunay triangulation $\mathcal{G} = (V, E)$, and compute the set of weights W of its edges.
2. Compute the minimum spanning tree (MST) $\mathcal{T}$ of $(\mathcal{G}, W)$.
3. Perform a depth-first search on $\mathcal{T}$ to compute with minimal redundancy the histograms of the N regions.

2.1. Graph construction

MST $\mathcal{T}$ results from a graph $\mathcal{G}$ representing the overlaps of the regions: each node of V corresponds to the center of a region and edges are assigned weights representing the overlap rate of their extremities. Intuitively, as all pairs of regions may overlap, $\mathcal{G}$ may be a complete weighted graph. However, in practice, distant regions are unlikely to overlap and their corresponding edges can be safely discarded. Doing so significantly speeds-up $\mathcal{T}$'s construction. Now, by setting $\mathcal{G} = (V, E)$ as the Delaunay triangulation of V (i.e., $\mathcal{G}$ is the set of triangles connecting the points of V such that their circumcircles do not contain any point in V), we discard precisely these unpromising edges. E thus contains $3N$ edges, each one linking some nodes $\mathbf{x}^p, \mathbf{x}^q$ that are the centers of regions R^p and R^q respectively. Weights W are defined as the distance between $\mathbf{x}^p$ and $\mathbf{x}^q$, given by norm L_1.

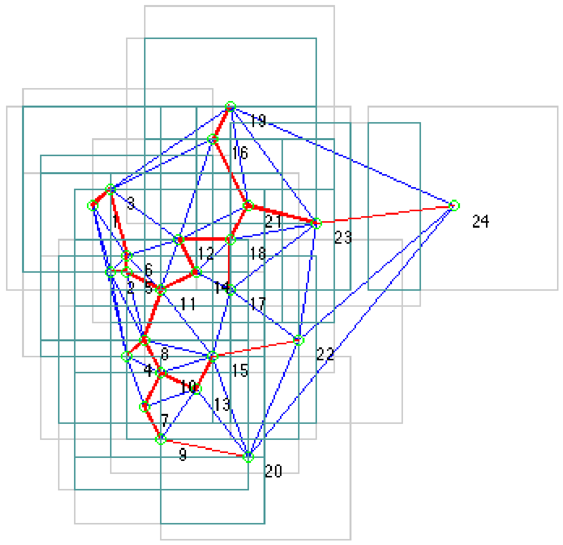

Fig. 2. Illustration of the proposed approach for $N = 30$ centers. A Delaunay triangulation is first constructed from center set V (in blue), on which Kruskal's algorithm is applied to give the minimum spanning tree, in red (bold red lines are those satisfying overlapping criterion c defined below).

2.2. Minimum spanning tree (MST)

Kruskal proposed in 1956 an algorithm to determine the MST $\mathcal{T}$ of a weighted graph $G = (V, E)$ [9]. Initially, $\mathcal{T}$ is set to $(V, \emptyset)$, i.e., a graph with no edge. The edges of E are ordered by increasing weight and are processed one by one in this order. Processing an edge $e \in E$ consists in adding it to $\mathcal{T}$ if it does not create any cycle in $\mathcal{T}$, else it is simply discarded. After processing all the edges in E, $\mathcal{T}$ is guaranteed to be a MST of G. Now, the key idea of our algorithm is that $\mathcal{T}$ contains the edges with the smallest weights, i.e., the pairs of the closest regions, those that should maximize overlaps. Therefore, to compute the histogram of a given node, we shall exploit those already computed in its neighbors to minimize redundant computations. This is detailed in the next subsection.

2.3. Graph traversal: a depth-first search

We apply a depth first search on $\mathcal{T}$ to determine the processing order of the regions that minimizes the number of operations necessary to compute their N histograms. For each edge $(\mathbf{x}^p, \mathbf{x}^q) \in \mathcal{T}$, we use an *overlapping criterion* to determine whether histogram H^q of region R^q should be updated from H^p or computed from scratch. Histogram H^q is updated from H^p using the idea in [3] illustrated in Fig. 1(b): keep the shared area (in gray), remove the pixels belonging only to R^p ($|R^p|^-$ subtractions) and add those only belonging to R^q ($|R^q|^+$ additions). This scheme is attractive when the number of operations necessary for updating H^q from H^p is lower than the number of operations $|R^q|$ necessary to compute the histogram of R^q from scratch (for example, this is not the case for Fig. 1(b)). The *overlapping criterion* is then given by:

$$c = \max\left(0; 1 - \frac{|R^p|^- + |R^q|^+}{|R^q|}\right) \quad (1)$$

If $c = 0$, H^q shall be computed from scratch, else it should be updated from H^p. Figure 2 shows an illustration for $N = 30$ centers (numbered): regions are gray rectangles, the Delaunay triangulation is drawn in blue, the MST in red, and edges satisfying the overlapping criterion are drawn in bold red.

Algorithm 1: Global algorithm

```
Input: set of regions $\mathbf{R} = \{R^1, \dots, R^N\}$
$V = \{\mathbf{x}^1, \dots, \mathbf{x}^N\} \leftarrow$ centers of $\{R^1, \dots, R^N\}$
$\mathcal{G} = (V, E) \leftarrow$ Delaunay triangulation of $V$
foreach $e_k = (\mathbf{x}^p, \mathbf{x}^q) \in E$ do
    $w_k \leftarrow \text{dist}_{L_1}(\mathbf{x}^p, \mathbf{x}^q)$
$W \leftarrow \{w_k : e_k \in E\}$
$\mathcal{T} \leftarrow$ Kruskal$(V, E, W)$
Choose any vertex $v$ in $\mathcal{T}$
$H^v \leftarrow$ compute the histogram of $R^v$ from scratch
depth_first_search$(\mathcal{T}, v, \mathbf{R})$ (see Algo. 2)
```

Algorithm 2: Depth first search algorithm

```
Input: tree $\mathcal{T}$, vertex $v$, a set of regions $\{R^1, \dots, R^N\}$
Label $v$ as explored
foreach neighbor $v_s$ of $v$ in $\mathcal{T}$ do
    if $v_s$ is not explored then
        if overlapping criterion$(R^v, R^{v_s}) \neq 0$ then
            $H^{v_s} \leftarrow$ compute histogram from $H^v$
        else
            $H^{v_s} \leftarrow$ compute histogram from scratch
        Label $v_s$ as explored
```

3. EXPERIMENTS

In this section, we experiment the efficiency of our Kruskal-based approach on the problem of computing N different histograms. We thus randomly generate N locations (x, y) corresponding to the centers of N non-rotated rectangular regions of a fixed size $w \times h$. The total area covered by all these regions is called the region of interest (ROI). To test our approach, we compare it against the *naive* one consisting in computing each histogram by scanning all the pixels of its region, and also against Integral histogram (IH). The latter first parses the ROI to generate an accumulator and, then, computes the N histograms using only 3 arithmetic operations on that accumulator. Different parameters affect the total computation times: the number N of histograms to compute, the number B of bins in the histograms, the size of the regions in which histograms are computed, and the size of the ROI.

Figure 3.(a) shows comparative computation times in function of the size of the ROI for computing $N = 250$ $B = 16$-bin histograms of 30×30-size regions. As we can see, unlike IH, the naive and the Kruskal-based approaches do not depend on the size of the ROI. Note that our approach induces the lowest computation times.

Figure 3.(b) shows comparative computation times in function of the size of the regions in which we compute histograms (here we consider square regions: $w = h$). Computation times increase quadratically with w in the naive approach, and increase linearly in the other approaches. However, they increase faster with IH: for $w > 40$, computation times are lower with our approach.

Figures 3.(c-d) show comparative computation times in function of the number N of histograms to compute. As shown in Figure 3.(a), the size of the ROI affects IH computation times. This explains why we test the influence of N on two different ROI sizes. For the smaller size (150×170), IH is better than our approach for $N > 1250$ histograms: this is well-known that IH is well-suited for computing numerous histograms. For a larger ROI (270×250), the Kruskal-based approach always induces lower computation times. This is due to the fact that the computation of IH's accumulator is very costly and depends directly on the size of the ROI.

Figures 3.(e-f) show comparative computation times in function of the number B of bins in the histograms. Here again, we test different values for N (number of histograms to compute). All regions have a size of 30×30, and that of the ROI is fixed. For $N = 500$, our approach induces lower computation times whatever the value of B (note that IH is the worst approach for $N = 500$: the computation of the accumulator is far too costly). For $N = 2000$, IH becomes better for $B \geq 128$. Our test have shown that IH becomes the best approach for higher values of N.

The attractive feature of our algorithm is its capacity to exploit spatial redundancies arising when regions overlap. We define the overlapping rate of a set S of regions as the number of pixels belonging to at least two regions in S divided by the number of pixels in the union of all the regions in S. Table 1 illustrates the behavior of our approach w.r.t. this rate. As can be expected, the overlapping rate has an impact on the computation times. Those can be reduced by a factor 5.

Table 1. Computation times t (in sec.) with our approach to compute $N = 200$ histograms of regions of size 20×20 depending on the total overlapping rate r of their regions.

r	0	0.25	0.5	0.75	1
t	0.11	0.057	0.043	0.027	0.024

As a conclusion to our tests, first note that our Kruskal-based approach depends neither on the quantification of the

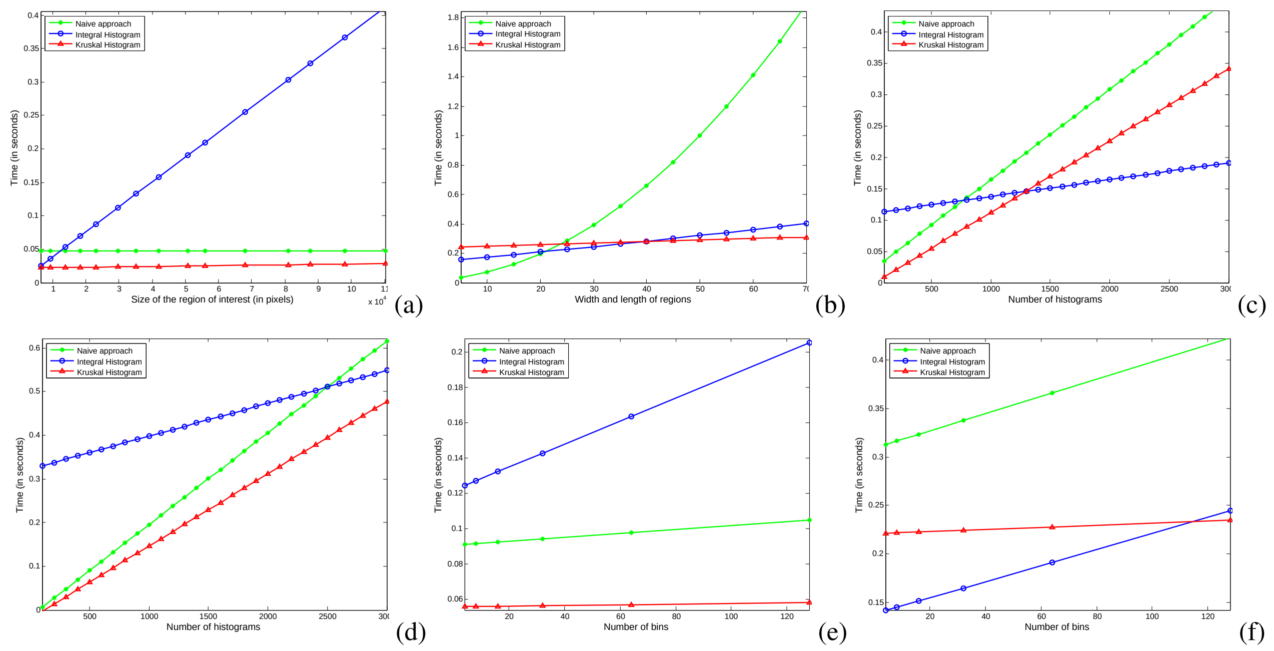

Fig. 3. Comparison of computation times depending on (a) the size of the ROI ($N = 250$) and (b) the size of the regions in which histograms are computed ($N = 2000$), (c) the number N of histograms to compute ($w = h = 30$, $B = 16$, ROI size: 150×170), (d) the number N of histograms to compute ($w = h = 30$, $B = 16$, ROI size: 270×250) (e) the number B of bins ($w = h = 30$, $N = 500$, ROI size: 150×170) and (f) the number of bins ($w = h = 30$, $N = 2000$, ROI size: 150×170).

histograms (B value) nor on the size of the ROI. Second, its computation times slowly increase with the size of the regions in which histograms are computed. In addition our approach is more effective than the other tested approaches when the regions are large. Finally, unlike IH, our approach is effective even when the number of histograms to compute is rather small. However, it is less adapted than IH when the number of histograms to compute is high (note that this also depends on the size of the regions in which histograms are determined as well as on the quantization of these histograms).

4. CONCLUSION

We have presented a new approach for fast histogram computation that exploits overlapping regions. The main idea consists in finding the order in which the histograms of these regions have to be processed to minimize the computation redundancies. We have shown that this approach is very efficient compared with the naive approach but also with integral histogram when the size of the regions or that of the global area they cover is large. In addition, our approach does not dependent on the quantization of the histograms and is adapted for the computation of a large number of histograms. This makes this method well-suited when large spaces need to be explored while keeping a fine description of regions. For future researches, we work on the adaptation of our approach on rotated rectangular regions.

5. REFERENCES

[1] N.J. Gordon, D.J. Salmond, and A.F.M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *IEE Proceedings of Radar and Signal Processing*, vol. 140, no. 2, pp. 107–113, 1993.

[2] P. Pérez, C. Hue, J. Vermaak, and M. Gangnet, "Color-based probabilistic tracking," in *ECCV*, London, UK, 2002, pp. 661–675.

[3] G.Y. Tang, G.J. Yang, and T.S. Huang, "A fast two-dimensional median filtering algorithm," in *IEEE Transactions on Acoustics, Speech and Signal Processing*, 1979, vol. 27, pp. 13–18.

[4] S. Perreault and P. Hebert, "Median filtering in constant time," *IEEE Transaction on Image Processing*, vol. 16, no. 9, pp. 2389–2394, 2007.

[5] M. Sizintsev, K.G. Derpanis, and A. Hogue, "Histogram-based search: A comparative study," *CVPR*, pp. 1–8, 2008.

[6] F. Porikli, "Integral histogram: A fast way to extract histograms in cartesian spaces," in *CVPR*, 2005, pp. 829–836.

[7] A. Adam, E. Rivlin, and I. Shimshoni, "Robust fragments-based tracking using the integral histogram," in *CVPR*, 2006, pp. 798–805.

[8] S. Dubuisson, "Tree-structured image difference for fast histogram and distance between histograms computation," *Pattern Recognition Letters*, vol. 32, no. 3, pp. 411–422, 2011.

[9] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, 1956.