Integral Histogram: A Fast Way to Extract Histograms in Cartesian Spaces

Fatih Porikli
Mitsubishi Electric Research Laboratories
fatih@merl.com

Abstract

We present a novel method, which we refer as an integral histogram, to compute the histograms of all possible target regions in a Cartesian data space. Our method has three distinct advantages: 1- It is computationally superior to the conventional approach. The integral histogram method makes it possible to employ even an exhaustive search process in real-time, which was impractical before. 2- It can be extended to higher data dimensions, uniform and non-uniform bin formations, and multiple target scales without sacrificing its computational advantages. 3- It enables the description of higher level histogram features. We exploit the spatial arrangement of data points, and recursively propagate an aggregated histogram by starting from the origin and traversing through the remaining points along either a scan-line or a wave-front. At each step, we update a single bin using the values of integral histogram at the previously visited neighboring data points. After the integral histogram is propagated, histogram of any target region can be computed easily by using simple arithmetic operations.

1. Introduction

A histogram is an array of numbers in which each element, bin, corresponds to the frequency of a range of values in the given data. For instance, each bin counts the number of pixels having the same color values in case of an image histogram. Thus, a histogram is a mapping from the set of data values to the set of non-negative real numbers. From a probabilistic point of view, the normalization of an histogram results in a function that is most akin to the probability density function of the data. It is possible to employ a histogram to answer the following questions: What kind of distribution do the data come from? What are the statistical properties of this distribution such as how spread out are the data? Are there outliers in the data? Histograms are among the most common features used in many computer vision tasks from object based retrieval [1], [2], to segmentation [3], [5] to detection [4], [6] to tracking [7].

Computational complexity is one major bottleneck of the histogram extraction and comparison based search tasks. It is obvious that only an exhaustive search can provide the global optimum. Although several sub-optimal techniques that are powered by gradient descent methods and application specific constraints have been developed to deliver accelerated alternatives to the basic exhaustive search, computer vision problems that rely on the optimal solutions, such as detection and tracking, still demand a theoretical breakthrough in histogram extraction as much as an powerful computers to crunch the numbers.

To address the computational requirements of detection tasks, we develop a fast method to compute histograms of all possible target regions in a given data. We take advantage of the spatial positioning of data points in a Cartesian coordinate system, and propagate an aggregated function, which we refer as the integral histogram, starting from an origin point and traversing through the remaining points along a scan-line. We iterate the integral histogram at the current point using the histograms of the previously processed neighboring data points. At each step, we increase the value of the bin that the current point fits into the bin's range. After the integral histogram is obtained for each data point, histograms of target regions can be computed easily by using the integral histogram values at the corner points of those regions without reconstructing a separate histogram for every single region. In a 2D data, such as an image, the integral histogram converts into the extraction of rectangular region histograms, which are computed by intersection of the integral histogram at the four corner points.

The integral histogram method has several advantages: First, it is computationally superior than the conventional approach. It is possible to execute even an exhaustive histogram search process in the data space, which was infeasible with conventional approaches. It can be extended to higher dimensions, histogram bin structures, and multiple scales without sacrificing its computational benefits. It enables description of advanced histogram features as illustrated in Fig. 1.

In the next section, we summarize the previous work. In section 3, we introduce the integral histogram formulation

Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)
1063-6919/05 $20.00 © 2005 **IEEE**

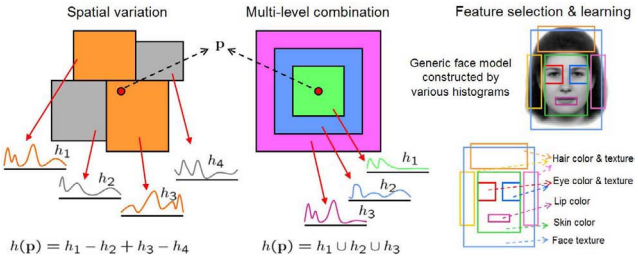

Figure 1. Advanced features, e.g. spatial arrangement or hierarchical fusion of the component histograms, can be easily computed using integral histogram for various tasks.

in detail. In section 4, we give a computational complexity analysis by considering different scenarios. In section 5, we present simulation results and discuss various aspects of the proposed method.

2. Previous Work

It is possible to calculate the sum of the values within rectangular regions in linear time without repeating the summation operator for each possible region [6]. A constant number of operation for each rectangular sum is needed to compute such sums over distinct rectangles many times. A cumulative image function is defined such that each element of this function holds the sum of all values to the left and above of the pixel including the value of the pixel itself. The cumulative image can be computed for all pixels with four arithmetic operations per pixel. Starting from the top left corner and traversing first to the right and then to the down, the value of the cumulative image at the current pixel equal is obtained by the addition of the left and the up pixel and subtraction of the upper left pixel's cumulative values. After the cumulative image is computed, the sum of image function in a rectangle can be computed with another four arithmetic operations with appropriate modifications at the border. Thus with a linear amount of computation, the sum of image function over any rectangle can be computed in linear time.

A conventional approach of measuring distances between a given histogram and histograms of all possible target regions is an exhaustive search. This process requires generation of histograms for the regions centered at every possible points. In case the search should be done at different scales, i.e. different target region sizes, the whole process should be repeated as many times as the number of scales. We give a pseudo-code of the conventional histogram in algorithm 2.1. To our knowledge, the conventional approach is the only solution (other than the presented integral histogram method) that guarantees to find the global optimum in histogram based search.

Algorithm 2.1: CONVENTIONAL(N, M, S, B)

for each *possible scale* $\in S$
do
  for each *possible point* $\in N$
  do
    for each *target point* $\in M$
    do
      Get current value
      Find corresponding bin
      Get bin value
      Increase bin value
    for each *bin* $\in B$
    do *Normalize*
    Compute histogram distance

3. Integral Histogram Formulation

Integral histogram is a recursive propagation method works in Cartesian spaces and it can be extended into any dimensional data space and any tensor representations. It is a superset of the cumulative image formulation mentioned in the previous section. To perform histogram comparison, we first generate an integral histogram by propagation, and then compute the histograms of target regions by intersection.

Suppose our function f is a defined in a d-dimensional real valued Cartesian space $\mathcal{R}^d$ such as $\mathbf{x} \rightarrow f(\mathbf{x})$ where $\mathbf{x} = [x_1, .., x_d]$ is a point in this space. This function maps to a k-dimensional tensor, i.e. $f([x_1, .., x_d]) = [y_1, .., y_k]$. Let assume the d-dimensional data space to be bounded within the range $N_1, ..., N_d$, i.e. $0 \leq x_i \leq N_i$.

3.1. Propagation

We define an integral histogram $H(\mathbf{x}^p, b)$ at a data point at the p^{th} order along a sequence of points $\mathbf{x}^0, \mathbf{x}^1, .., \mathbf{x}^p$ such as

$$H(\mathbf{x}^p, b) = \bigcup_{j=0}^{p} Q(f(\mathbf{x}^j)) \tag{1}$$

where $Q(.)$ is the corresponding bin of the current point, and $\cup$ is the union operator that is defined as follows: the value of the bin b of $H(\mathbf{x}^p, b)$ is equal to the sum of the previously visited points's histogram bin values, that is the sum of all $Q(f(\mathbf{x}^j))$ while $j < p$. In other words, $H(\mathbf{x}^p, b)$ is the histogram of the region between the origin and current point; $0 \leq x_1^j \leq x_1^p$, $0 \leq x_2^j \leq x_2^p$, ..., etc. Note that, $H(\mathbf{x}^N, b)$ is equal to the histogram of all data points since $\mathbf{x}^N = [N_1, .., N_d]$ is the last point in the space. Therefore, the integral histogram can be written recursively as

$$H(\mathbf{x}^j, b) = H(\mathbf{x}^{j-1}, b) \cup Q(f(\mathbf{x}^j)) \tag{2}$$

using the initial condition $H(0, b) = 0$, which means all the bins are empty at the origin.

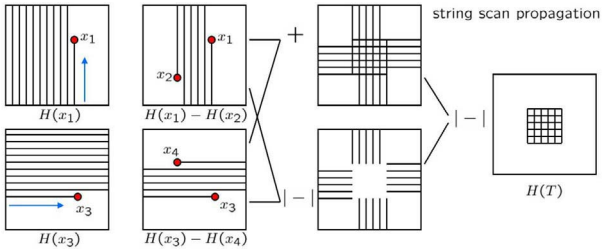

Figure 2. Propagation of integral histogram by string scan.

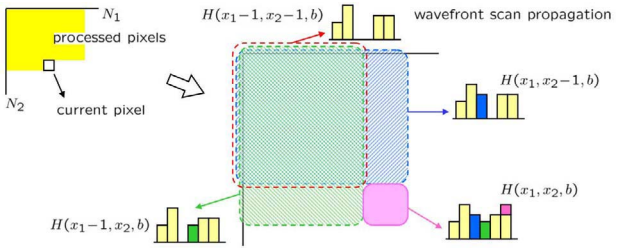

Figure 3. Propagation of integral histogram by wavefront scan. Yellow indicates already traversed points. At each step, the current integral histogram is obtained from the integral histogram values of the three neighbors, and the bin that corresponds to current point's value is increased by one.

The are different scanning and propagation approaches; here we present two of them. One is a string scan method that covers the data space along each dimension e.g. from left to right and top to bottom for an image data. The integral histogram at the current point is obtained by copying the previous values and increasing the corresponding bin with respect to the current value of the point. The string scan requires only update at each step of propagation. However, d string scans at different dimensions should be performed to obtain the histogram of a polytope region in a d-dimensional data as illustrated in Fig. 2.

It is also possible to scan points using an active sets of points, i.e. a wavefront. The wavefront scan requires updating the integral histogram for such data points that their left, upper, and upper-left neighbors are already scanned in case of an image data. The integral histogram at a point is obtained by three arithmetic operations for each bin of using the integral histogram values of the three neighbors as shown in Fig. 3. The integral histogram values of the previous point is copied to the current point before the propagation. Either the updated bin is copied to all of the remaining points' bins (a total of $0.5(N^2 - N)$ copy operations), or all the previous bins are copied to the current bins (BN operations), which can be done by fast hardware-level memory copy functions or by pointer tables.

3.2. Intersection

The histogram of a target region T can be computed using the wavefront propagated integral histogram values at the boundary points of the region. In a Cartesian space, the target region corresponds to a polytope enclosed by a finite number of hyperplanes such as $x_1^- \leq x_1 \leq x_1^+, x_2^- \leq x_2 \leq x_2^+, .., x_d^- \leq x_d \leq x_d^+$. The boundary points are $\mathbf{x}_l^r$ where their indices in each dimension have r number of x^- coordinates and $d-r$ number of x^+ coordinates. Note that, for a fixed r there exist C_r^d such combinations, which is a binomial coefficient. In other words, for $r = 0$ there is only one point $[x_1^+, x_2^+, .., x_d^+]$. For $r = 1$ there are d points $[x_1^-, x_2^+, .., x_d^+]$, $[x_1^+, x_2^-, .., x_d^+]$,...,$[x_1^+, x_2^+, .., x_d^-]$, and so on. Then, the histogram is simply obtained as

$$h(T, b) = \sum_{r=0}^{d} (-1)^r \sum_{l=1}^{C_r^d} H(\mathbf{x}_l^r, b). \quad (3)$$

This assigns the histogram bins of the current point by using the intersection of the bins of the three previous histograms.

In case of an $N_1 \times N_2$ gray level image, our parameters are $d = 2$, $k = 1$, and a wavefront scan from upper left point the propagation can be written as

$$H(x_1, x_2, b) = H(x_1-1, x_2, b) + H(x_1, x_2-1, b) - H(x_1-1, x_2-1, b) + Q(f(x_1, x_2)). \quad (4)$$

and the intersection becomes $h(T, b) = H(p_1^+, p_2^+, b) - H(p_1^-, p_2^+, b) - H(p_1^+, p_2^-, b) + H(p_1^-, p_2^-, b)$.

As opposed to the conventional histogram computation, the integral histogram method does not repeat the histogram extraction for each possible region as given in the pseudo-code below:

Algorithm 3.1: INTEGRAL HISTOGRAM(N, S, B)

```
for each possible point ∈ N
do { for each bin ∈ B
     do {Propagate integral histogram
     Get current value
     Find bin
     Get bin value
     Increase bin value
for each possible scale ∈ S
do { for each possible point ∈ N
     do { for each bin ∈ B
          do {Compute intersection
              Normalize
          Compute histogram distance
```

	A	B	C	D	E
Integer addition	1	1	1	1	1
Integer multiply	4	4	1.2	24	4
Integer divide	6	36	4.4	-	75
Floating-point addition	20	3	1	4.2	4
Floating-point multiply	20	5	1.2	113	4
Floating-point divide	20	38	1.2	-	100
Type conversion	20	-	-	-	105
Bit-wise shift	1	-	-	-	2

Table 1. Column-A is the relative cost of the basic processor operators as given in [8]. Column-B is the cost of the operators executed on a P4 processor that uses streaming SIMD and Prescott arithmetic operations [10]. Column-C is the corresponding costs on a P3 MMX processor [11]. Column-D is the relative costs on a P4 working running C++ compiler [9]. We also did our own experiments to determine the relative costs (column-E).

4. Complexity Analysis

We performed a computational complexity analysis in terms of the relative cost of processor operations, which is usually measured against the cost of an integer addition operation. Relative costs of several operations reported in the literature as well as our own observations are presented in the Table 1.

Since the cost of the array indexing becomes comparable especially for the higher dimensional data, we also make an assessment of the indexing operators. In [8], it is explained that an ordinary indexing for an d-dimensional array requires d additions, d–1 multiplications, and d logical operators, which has a total relative cost of d+4(d–1)+1 = 6d–4. By using a look-up table of pointers, the multiplications can be replaced by d–1 pointer referencing. However, we found that the cost of an d-dimensional array indexing is approximately $4d+3(d-1) = 7d-3$ in our experiments.

We assume the input data is a d-dimensional array with k-dimensional tensors. The histograms are k-dimensional with B identical size bins assigned for each dimension, and the bin size is also an integer number. The target region size is $M_1 \times .. \times M_d$. Most problems also require extraction of histograms at different scales S_s where $s = 1, .., d$. The type of the input data, i.e. whether it is integer or floating point, changes the computational load. The below analysis can be extended to fixed point operations as well.

4.1. Integer Data

Suppose the input data has integer valued tensors. The conventional histogram matching algorithm requires these main tasks before comparing histograms:

- Get current values: 1 d-dimensional array indexing and k additions,
- Find bin: k integer divisions (or floating point multiplication and float-to-integer conversion),
- Get bin value: k-dimensional array indexing,
- Increase bin value: 1 integer addition,
- Normalize: B^k floating point multiplication.

Note that, for different region sizes, the above computation should be repeated. In terms of the relative cost, the conventional algorithm requires $7d-3+k$ operations for getting the current values in the d-dimensional input tensor, $75k$ operations to compute the corresponding bin indices, 1 operation (for 1 addition) to increase the bin value. Computing bin indices can be done by a floating-point multiplication and then float-to-integer conversion, however the cost of this option ($109k$) is higher than the division itself ($75k$). After all the $M_1 \times .. \times M_d$ points in the target region are processed, the histogram bins are normalized with the number of points, which requires B^k floating point multiplications, thus $4B^k$ operations in terms of the relative cost. Note that the previous computations are repeated for each of the $N_1 \times .. \times N_d$ histograms matches. Then, the total number of operations needed for all candidates becomes

$$\left[(7d + 76k - 2) \prod_{j}^{d} M_j + 4B^k \right] \prod_{i}^{d} N_i \prod_{s}^{d} S_s \qquad (5)$$

On the other hand, the integral histogram method needs

- Propagate integral: 3 k-dimensional array indexing and $2k$ integer additions,
- Get current values: 1 d-dimensional array indexing and k additions,
- Find bin: k integer divisions (or floating point multiplication and float-to-integer conversion),
- Get bin value: k-dimensional array indexing,
- Increase bin value: 1 integer addition,
- Compute intersection: 4 k-dimensional array indexing and $3k$ integer additions,
- Normalize: B^k floating point multiplications.

Thus, the propagation takes $3(7k-3)+2k = 23k-9$ operations in addition to the cost of getting the current value of the tensor values ($7d-3+k$), finding the indices of the corresponding bin ($75k$), and accumulating the obtained bin value (1), which is repeated for all points in the data space. Then, we find that $(7d + 99k - 11) \prod_i^d N_i$ operations are required to construct the integral histogram. We compute the histogram intersection using $4(7k-3)+3k = 31k-12$

operations, and normalize the result using B^k floating point divisions ($4B^k$ operations) for each histogram. Then, the cost of extraction of all histograms at all possible scales is

$$\left[7d + 99k - 11 + (31k - 12 + 4B^k)\prod_s^d S_s\right]\prod_i^d N_i \quad (6)$$

Of course, both methods compute histogram distances using the given metric in addition to the above costs.

We define a ratio of the computational load of the conventional approach versus the integral histogram method;

$$r = \frac{[(7d + 76k - 2)\prod_j^d M_j + 4B^k]\prod_s^d S_s}{7d + 99k - 11 + (31k - 12 + 4B^k)\prod_s^d S_s} \quad (7)$$

4.2. Floating Point Data

Use of floating point data increases the cost of the divisions in the computation of the bin indices. The cost increases from $75k$ for each point to $100k$. The bin value increment cost becomes 4, which was 1 before. The total cost for the conventional approach becomes

$$\left[(7d + 101k + 1)\prod_j^d M_j + 4B^k\right]\prod_i^d N_i \prod_s^d S_s \quad (8)$$

For the integral histogram method, the complexity of the step for finding bin indices increases to $100k$. In the propagation stage, the cost of additions rises from $2k$ to $8k$. In the intersection computation, the cost becomes $4(7k - 3) + 12k = 40k - 12$. The total cost becomes

$$\left[7d + 130k - 11(40k - 12 + 4B^k)\prod_s^d S_s\right]\prod_i^d N_i \quad (9)$$

4.3. Power-of-2 Bin Sizes

Note that further optimizations on the both methods is possible by using a bin size that is a power of 2. Using bit-wise shift operator, a division operator can be achieved with a fraction of the cost. For instance, instead of dividing by 64, we can shift the number 6 bits to the right. The computation of the bin indices drops from $75k$ to $2k$ (on average) depending on the amount of the shift. Then the total number of operations for integer data using the conventional approach becomes

$$\left[(7d + 3k - 2)\prod_j^d M_j + 4B^k\right]\prod_i^d N_i \prod_s^d S_s \quad (10)$$

The integral histogram also gains using the bin sizes that are values of 2. The total cost drops to

$$\left[31k + 7d + 1 + (43k + 1 + 100B^k)\prod_s^d S_s\right]\prod_i^d N_i \quad (11)$$

4.4. Matching Without Normalization

In certain applications, the target object is searched in its original size without a scaling, or with scaling factors of half sizes that correspond to downsampling by powers of 2, i.e. half size, quarter size, etc. In such cases, further computational reduction is possible in both methods since no histogram normalization is needed for the same size matches.

For a scaling factor of 2^{-s}, where $s = 0$ stand for no scaling, $s \geq 1$ for downsizing, the necessary computations of the conventional approach with integer data becomes

$$\left[(7d + 35k + 4)\prod_j^d M_j + 5(1 - \delta(s))B^k\right]\prod_i^d N_i \quad (12)$$

And the integral histogram performs in

$$\left[7d + 26k - 11 + (31k - 12 + 4B^k)\prod_s^d S_s\right]\prod_i^d N_i \quad (13)$$

Note that, in addition to above costs, the conventional approach has another important disadvantage. After each computation, it needs the histogram array values to be destroyed, which creates additional overhead.

5. Examples

5.1. 1D Case: Time Series

For an ordinary 1D data such as time series with a given length M and a histogram with a total bin number B, a target size range up to S points, the parameters of the above analysis become $d = 1$ and $k = 1$. We obtain the ratio as

$$r_1 = \frac{(81M + 4B)S}{95 + (19 + 4B)S} \quad (14)$$

We present the computational ratio results for 1-D data in Fig. 4 (1^{st} row). The different graphs in the first column represents the different target sizes plotted against the different number of bins in the histogram. The vertical axis shows the amount of computational savings. As visible, the integral histogram improves the processing time up to the 3.5×10^4 times over the conventional method. For instance, a common task that requires searching a pattern which contains 10^4 points using a 32-bins histogram can be employed 3,347 times faster than the conventional method.

5.2. 2D Case: Gray Level Images

For a $M_1 \times M_2$ gray level image and a search region size range S_1, S_2, the parameters of the above analysis become $d = 2$ and $k = 1$, and the ratio is

$$r_2 = \frac{[88M_1M_2 + 4B]S_1S_2}{102 + (50 + 4B)S_1S_2} \quad (15)$$

2-D data is very common in most vision problems from gray-level surveillance video to mono-chrome aerial imagery. For instance, our problem may involve finding a 64×64 target pattern in 3 different resolutions using a 16-bins histogram. The integral histogram method hunts for these patterns $2,435$ times faster. In Fig. 4 (2^{nd} row), we give the comparison results, which show the integral histogram performs up to 6×10^4 times faster computations.

5.3. 2D Case with 3D Tensors: Color Images

For a color image with a 3D histogram (assuming each point has 3 color values in a tensor form), the parameters become $d = 2$ and $k = 3$. Assuming we are searching for a template at S_1, S_2 scales, the ratio becomes

$$r_3 = \frac{[240M_1M_2 + 4B^3]S_1S_2}{300 + (81 + 4B^3)S_1S_2} \tag{16}$$

In Fig. 4 (3^{rd} row), we present the computational savings for a color image (2-D data, 3-D histograms) search. Even for a regular model matching task that searches a 100×100 object model in 20 scales using histograms for each color channel coded in 4-bits (16-bins), the process is accelerated 146 times. As shown in the graphs, the savings can go up to 7×10^5 depending on the number of bins and target size.

5.4. 3D Case: Volumetric Data

A volumetric data on the other hand have $d = 3$ and $k = 1$. Searching in higher dimensional spaces is essential in feature selection and classification problems. The corresponding ratio is obtained as

$$r_4 = \frac{[95M_1M_2M_3 + 4B]S_1S_2S_3}{109 + (81 + 4B)S_1S_2S_3}. \tag{17}$$

Integral histogram method becomes more advantageous in higher dimensions as shown in Fig. 4 (4^{th} row). The savings can reach up to 15×10^7. For a $10^3 \times 10^3 \times 10^3$ target volume being searched in its original size ($S = 1$) using a 100-bins histogram, we can achieve 1.6×10^8 times improvement.

5.5. Object Detection Results

Figures 5-6 show detection results of given patterns using histogram features. In the traffic sign detection example, we search for the target object using a 2^{15}-bins color histogram. Although the conventional approach and integral histogram give the very same similarity map, the integral histogram method runs in 63msecs, however, the conventional approach requires 2 minutes on a 3.2Ghz P4. The integral histogram method is not limited to color and intensity histograms. In texture detection example, as given in Fig. 6, we use a 2^4-bins histogram of gradient orientation. To get the same results with the integral histogram

Figure 5. Object detection using a 2^{15}-bins color histogram. The computed similarity map is same as the conventional approach; however the integral histogram method runs in 63msecs although the conventional exhaustive search takes approximately 2 minutes for 100 scales on a 3.2Ghz P4.

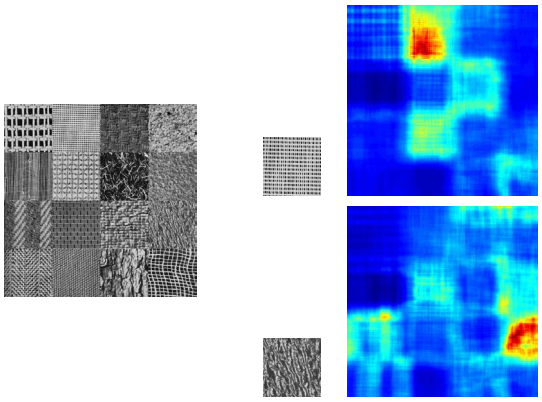

Figure 6. Texture detection using a 2^4-bins gradient orientation histogram. The integral histogram takes 88msecs, the conventional method requires more than 5 minutes to get the same result.

that takes 88 msecs, the conventional method requires more than 5 minutes of processing time. Note that, even such a simple histogram provides sufficient information for texture segmentation, and it is possible to combine histogram to define higher level features such as Haar wavelets, etc.

Note that, the integral histogram based search can be accelerated further by using application specific constraints as it is often employed for the conventional approach.

5.6. Tracking Examples

We simulated the integral histogram method to track objects between the consecutive video frames. After initialization of an object, we compute the color histogram similarity scores between the original histogram and the histograms of the object windows centered around every pixels. Note that, such a similarity computation would be very slow using the conventional approach. We compare our simple tracking adaptation with a gradient descent based method known as mean-shift [7]. Mean-shift evaluates the histogram similarity (in most cases using Bhattacharya distance) only within

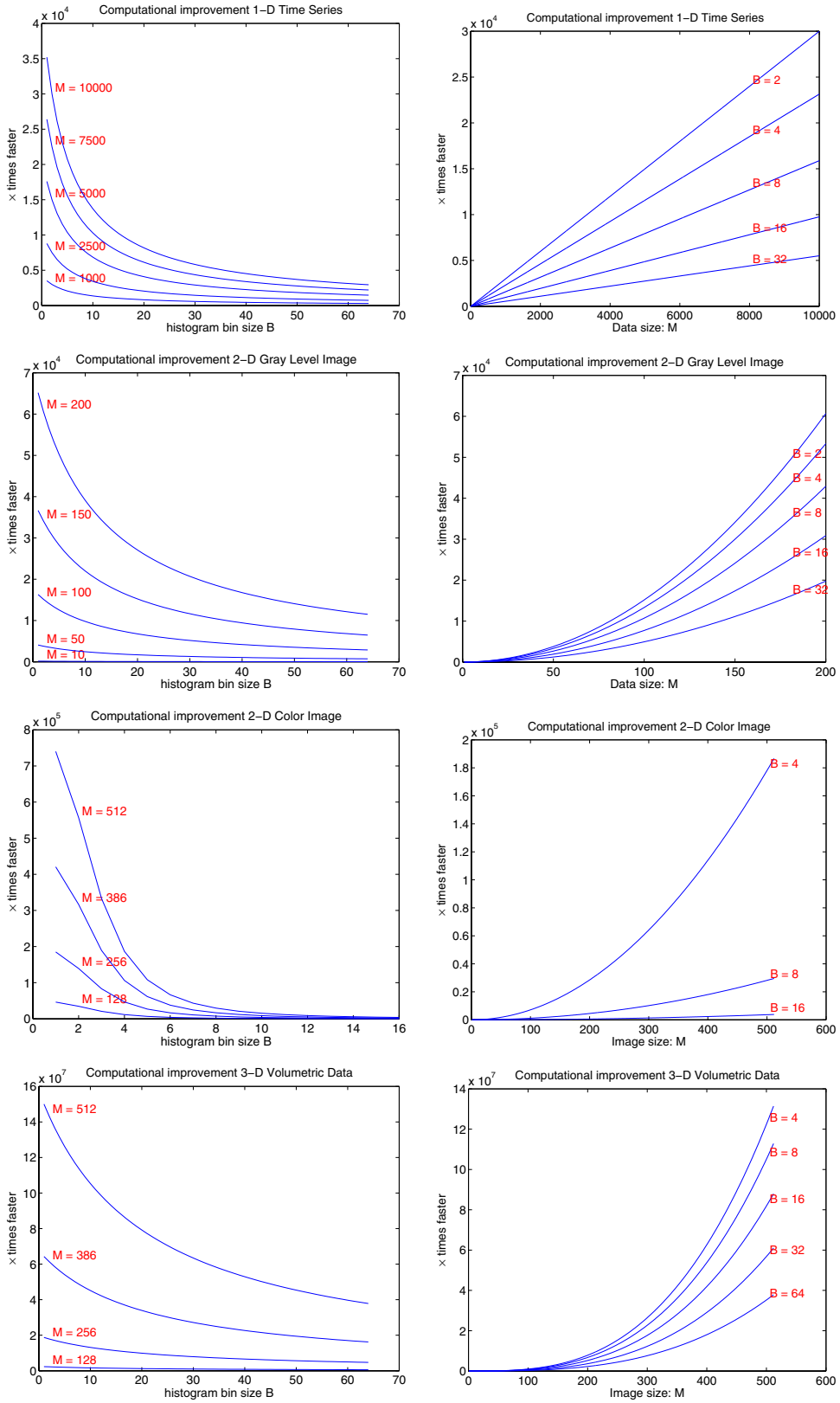

Figure 4. Computational reduction in comparison to the conventional method. The integral histogram method is *as many times as* faster than the existing approach for different type of problems explained in Section 5.

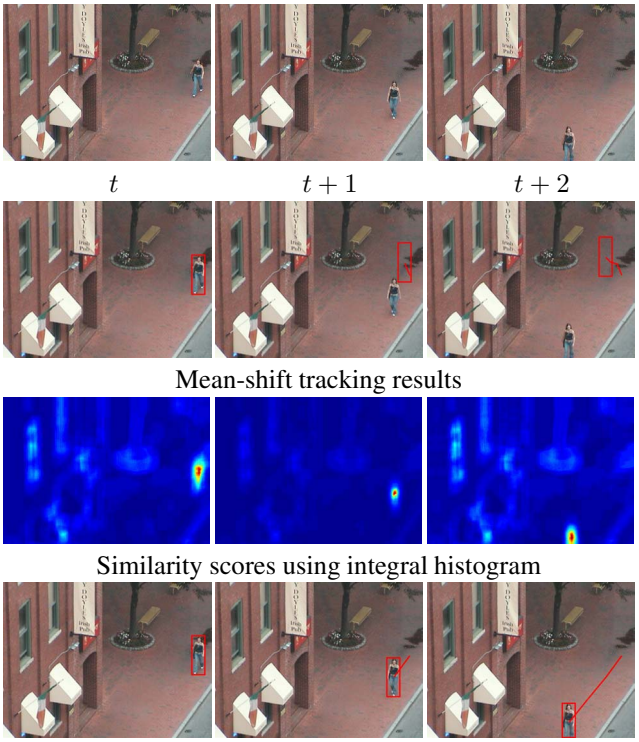

Figure 7. Object moves fast and there is no overlap between the consecutive frames. Although mean-shift needs only 15msecs on average, it may fail if the relocation of the object is large and there is no overlap of boxes. Integral histogram method can find the correct position in $\tilde{5}$5msecs regardless of the overlap.

its original kernel, that is the window of the object. Therefore, it is computationally feasible for real-time applications. For an object size shown in Fig. 7, the mean-shift iterations using 16-bins histograms for each color channel takes only 15 msecs on average depending the number of iterations (on 3.2Ghz P4). However, mean-shift owns its speed to the fact that it only evaluates the similarity within a limited search region. As a result, for the cases in which object relocation is large and there is no overlap between the object windows in the consecutive frames, it is bounded to fail as shown in the figures.

The integral histogram enables us to compute similarities all over the image plane in a relatively constant small amount of time (55msecs), thus we can track accurately fast objects even in high frame sampling rates that cause significant relocation of the objects.

6. Discussion

We present a novel and computationally very fast method to compute the histograms of all possible regions in a Cartesian space. The integral histogram provides not a suboptimal or a partial solution, but an optimum and complete solution for the histogram based search problems.

Our experiments with different number of bins, data dimensions, and data structures confirm that the integral histogram method drastically decreases the amount of computations needed to obtain a multitude of histograms, thus, it significantly improves the speed of search algorithms based on histogram comparison.

In addition, the integral histogram enables construction of advanced histogram features for further feature selection and classification purposes. It can be extended easily to higher dimensional data spaces and other tensor representations.

Several computer vision tasks such as video object detection and tracking where the real-time requirement was a bottleneck up to now will benefit from the integral histogram method.

References

[1] J. Huang, S. Kumar, M. Mitra, W.J. Zhu, and R. Zabih, "Image indexing using color correlograms", *In Proceedings of CVPR*, 1997.

[2] C. Carson, M. Thomas, S. Belongie, J.M. Hellerstein, and J. Malik, "Blobworld: A system for region-based image indexing and retrieval", *In Proceedings of ICVS*, 1999.

[3] D. A. Forsyth and J. Ponce. "Computer Vision: A Modern Approach", *Prentice Hall*, 2002.

[4] C. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection, *In Proceedings of ICCV*, 1998.

[5] S. Ruiz-Correa, L. G. Shapiro, and M. Meila, "A new paradigm for recognizing 3-D object shapes from range data", *In Proceedings of CVPR*, 2003.

[6] P. Viola and M. Jones, "Robust real-time face detection", *In Proceedings of ICCV*, page II: 747, 2001.

[7] D. Comaniciu, V. Ramesh, and P. Meer, "Real-time tracking of non-rigid objects using mean shift", *In Proceedings of CVPR*, 2000.

[8] S. Oualline, "Practical C++ programming", *O'Reilly & Associates*, ISBN: 1-56592-139-9, 1995.

[9] J. Mathew, P. Coddington and K. Hawick, "Analysis and development of java grande benchmarks", *In Proceedings of ACM*, 1999.

[10] R. Bryant and D. O'Hallaron, "Computer systems: a programmer's perspective", *Prentice Hall*, ISBN 0-13-034074-1, 2003.

[11] Y. Moon, F. Luk, H.C. Ho, T.Y. Tang, K. Chan, C. Leung, "Fixed-point arithmetic for mobile devices; a fingerprint verification case study", *In Proceedings of SPIE*, 2002.