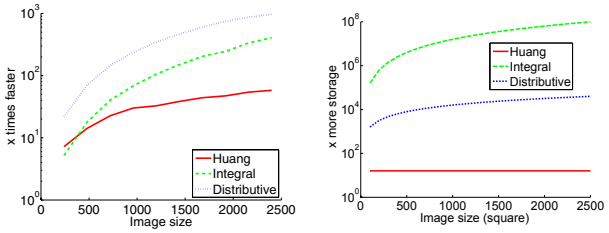
HISTOGRAM-BASED SEARCH: A COMPARATIVE STUDY

Mikhail Sizintsev, Konstantinos G. Derpanis
Department of Computer Science and Engineering
York University
Toronto, ON, Canada
{sizints,kosta}@cse.yorku.ca

Andrew Hogue
Faculty of Business and Information Technology
University of Ontario Institute of Technology
Oshawa, ON, Canada
andrew.hogue@uoit.ca

Abstract

Histograms represent a popular means for feature representation. This paper is concerned with the problem of exhaustive histogram-based image search. Several standard histogram construction methods are explored, including the conventional approach, Huang's method, and the state-of-the-art *integral histogram*. In addition, we present a novel multi-scale histogram-based search algorithm, termed the *distributive histogram*, that can be evaluated exhaustively in a fast and memory efficient manner. An extensive systematic empirical evaluation is presented that explores the computational and storage consequences of altering the search image and histogram bin sizes. Experiments reveal up to an eight-fold decrease in computation time and hundreds- to thousands-fold decrease of memory use of the proposed *distributive histogram* in comparison to the integral histogram. Finally, we conclude with a discussion on the relative merits between the various approaches considered in the paper.



(a) (b)

Fig. 1. Computational improvements (a) and storage requirements (b) of the various histogram methods compared to the conventional approach. The target size is set at 5% of the image size and 16-bin histograms were used in this experiment. Note the logarithmic scale for the vertical axes. Decreases in computation times among the methods are accompanied by increases in memory requirements. The *distributive histogram* proposed in this paper is not only the fastest of the alternative approaches, but represents the best speed-memory trade-off (see Section 5 for details).

1. Introduction

The histogram represents a popular feature representation in computer vision. Example applications include: object detection (e.g., [21, 9]), human detection (e.g., [8]), texture analysis (e.g., [13, 14]) and tracking (e.g., [2, 6, 5]). Histograms encode the distribution of spatially unordered image measurements in a region. More formally, a histogram is defined as an array of numbers, where each element (termed *bin*) corresponds to the frequency count of the range of values (e.g., image intensity, colour, gradient orientation, etc.) in the given image or subset. From a probabilistic viewpoint, the normalized histogram can be viewed as a probability distribution function. In the case of intensity and colour-based histograms, these histograms exhibit invariance to translation, in-plane rotation, and change slowly with out of plane rotations, object distance change and occlusion. Additionally, lighting invariance may be realized by transforming the input image using a suitable transform prior to histogram construction (e.g., normalized RGB, HSV, YUV colour spaces).

The traditional major impediment in the use of histograms in timely image search or real-time tracking applications is the computational complexity in the construction of the histogram. Generally, to speed up image search computations, common approaches include: performing localized searches (e.g., *mean-shift* [6] and *particle filtering* [15]), restricting computations to low resolution instances of the input imagery or pyramid methods (e.g., [4]), terminating the similarity computation prematurely if the result does not meet a threshold [1], and the use of specialized parallel computing hardware, such as FPGA (e.g., [12]), GPU (e.g., [10]) and dedicated hardware (e.g., [3]).

This paper is concerned with a comparative study between common histogram construction methods for exhaustive image search. In addition, building upon recent advances in median filtering, we present a new multi-scale exhaustive histogram-based search algorithm – the *distributive histogram*. Figure 1 presents a snapshot of our results where we compared four methods including the aforementioned new method, which are described in detail in Sec. 2 and 3. Note that we analyze both computational speed and storage requirements as both criteria are important in practice. The analysis shows a vivid trend for speed-memory tradeoff and one may conclude that the proposed new method is not only the fastest, but also entails low memory burden.

978-1-4244-2243-2/08/$25.00 ©2008 IEEE

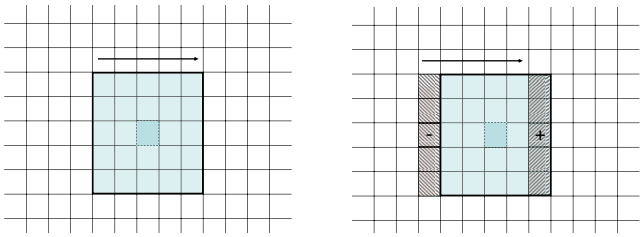


Fig. 2. Common approaches for histogram-based search. (a) conventional approach reinitializes histogram at each position ($O(r^2)$ per pixel cost) and (b) Huang's algorithm updates the kernel histogram by adding the rightmost column histogram and subtracting the leftmost ($O(r)$ per pixel cost).

The organization of this paper is as follows. In Sec. 2, a summary of the standard histogram constructions is presented. In Sec. 3, the distributive histogram formulation is introduced. In Sec. 4, a comparative computational (time) and memory (space) analysis of the histogram construction methods of concern is presented. In Sec. 5, systematic empirical evaluation of the computational and storage differences between the approaches is presented. Finally, Sec. 6 discusses the relative merits between the approaches.

2. Common histogram constructions

The conventional approach for exhaustive histogram-based search entails the construction of histograms centred at each possible position in the image (see Fig. 2(a)). The per pixel cost is $O(r^2)$, where r denotes the diameter of the target template; for ease of exposition we assume that the target is square. Although the conventional approach guarantees the localization of the global optimum, it is at the expense of significant computational effort that limits its usefulness in real-time or timely applications.

The state-of-the-art *integral histogram* [17, 14] represents a radical departure for computing histograms from the conventional approach. It is based on an extension of the *integral image* [22] (termed the *summed area table* in the graphics literature [7]). The integral image corresponds to a cumulative image function that is defined such that each element of this function contains the sum of all values to the left and above the pixel including the value of the pixel itself (assuming that the coordinate system is located at the top left corner of the image with x running to the right and y down). As pointed out in [17], the integral image can be computed in several ways. Note that care must be taken to avoid overflow errors. Given the integral image, a rectangular sum at any position and scale may be computed by three arithmetic operations. In [17, 14], the authors compute a separate integral image for each bin of the histogram. In the exhaustive search step, the value of a histogram bin at arbitrary histogram positions and radii can be computed by three arithmetic operations applied to the integral image of the respective bin.

In [17], a theoretical computational analysis of the integral histogram is presented. The conclusion drawn is that the integral histogram is computationally superior to the conventional approach. Empirically, they show that the integral histogram is capable of attaining real-time performance for exhaustive image search. At the same time, this approach also imposes substantial memory requirements, which limits its practical applicability, as will be shown in later sections.

In order to find a better histogram-based search method, let us turn back to the conventional approach and consider its direct descendants. Huang *et al*. [11] observed that there was a great amount of redundancy in computing the histograms of adjacent windows with the conventional method. Rather than recomputing the histograms at each image position, [11] proposed retaining only those histogram values in the intersecting region of the current histogram window and previous histogram window and adding the new histogram values in the adjacent column (see Fig. 2(b)). This approach realizes an $O(r)$ per pixel cost, where r is the target diameter.

Huang et al.'s approach addresses the redundant histogram calculations along a row but neglects the redundancies among rows. With the ultimate goal of median/bilateral filtering, Weiss [23] proposed an approach that accounted simultaneously for both sets of redundancies. Weiss' proposal realizes an $O(\log r)$ per pixel cost. More recently, Perreault and Hebert [16] proposed a simplification of Weiss' scheme that attains an $O(1)$ per pixel cost in the context of median filtering and computing rank order statistics. Given the conceptual simplicity and better potential for hardware implementation of Perreault and Hebert's approach over that of Weiss', in Sec. 3 we build upon the former's contribution with the goal of histogram-based image search (rather than image processing); we term this the *distributive histogram* approach. We will show that this method offers substantial performance improvement over all previous methods, while enjoying modest memory requirements. Additionally, the novelties of our extensions to the approach in [16], include: (1) accommodation to multi-scale processing, (2) construction of "advanced" histogram-based features and (3) extending the basic paradigm beyond histogram computations to accommodate regional statistics, such as the mean and covariance.

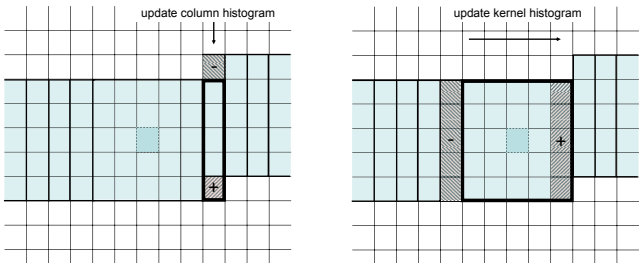
3. Distributive histogram approach

The main histogram property utilized in the construction of the distributive histogram is that of distributivity. Histogram distributivity means that for disjoint image regions A and B:

$$H(A \cup B) = H(A) + H(B), \tag{1}$$

where $H(\cdot)$ denotes the local histogram.

Figure 3 illustrates the basic idea behind the construction of the distributive histogram. For each column in the image, one histogram is maintained, which accumulates the bin weights of the r adjacent pixels. Note that the bin quantization can be of a uniform or non-uniform nature. The kernel



(a) (b)

Fig. 3. Distributive histogram construction. (a) the column to the right is updated ("moved down one row") by adding and subtracting a pixel and (b) the kernel histogram is updated by subtracting the leftmost histogram and adding the newly modified right column. Adapted from [16].

histogram is computed by summing r adjacent column histograms (following the distributivity property). As the kernel histogram sweeps to the right, the column histograms are added and subtracted as in Huang's method. The difference lies in the fact that the column histograms need not be fully recomputed each time since they are partially given in the previous row. As the histogram proceeds along the row, each rightmost column histogram encountered is updated by subtracting the topmost pixel and adding a single pixel below it, *i.e*. column histograms move downward (see Fig. 3).

For the case of histogram-based image search, the similarity of the kernel histogram at each image point is compared to that of the target model. Common histogram similarity measures in the literature include: $L1$ *norm*, $L2$ *norm*, *histogram intersection*, *Bhattacharrya distance* and χ^2*-statistic*. Also, one can optionally normalize the kernel histogram to a uniform sum in order to compensate for objects at different scales. The pseudo-code for the distributive histogram approach is given in Algorithm 1.

Algorithm 1 Distributive histogram approach.

Input: Image $I_{\text{input}}(x, y)$ of size $m \times n$, kernel radius r' (half-diameter $r' = r/2$) and model histogram H_{model}
Output: Similarity image $I_{\text{output}}(x, y)$ of size $m \times n$
Initialize kernel histogram H *and column histograms* $h(\alpha)$, $\alpha = 1, \ldots, m$
for $y = 1$ to n **do**
  for $x = 1$ to m **do**
    Remove $I_{\text{input}}(x + r', y - r' - 1)$ from $h(x + r)$
    Add $I_{\text{input}}(x + r', y + r')$ to $h(x + r)$
    $H \leftarrow H + h(x + r') - h(x - r' - 1)$
    Normalize (optional)
    $I_{\text{output}}(x, y) \leftarrow$ *Compute histogram similarity*
  end for
end for

Next, let us consider the computational complexity of the approach. Updating the kernel and column histograms is an $O(1)$ per pixel computation. Since the histogram size is fixed and independent of the target size, the similarity computation is $O(1)$ per pixel. Initialization is $O(r)$ per each column histogram; however, the *amortized* cost per pixel is still $O(1)$ when image size is proportional to the kernel size, $r = O(n)$, *i.e*. $\frac{O(1)n^2 + O(r)n}{n^2} = O(1) + O(n)/n = O(1)$.

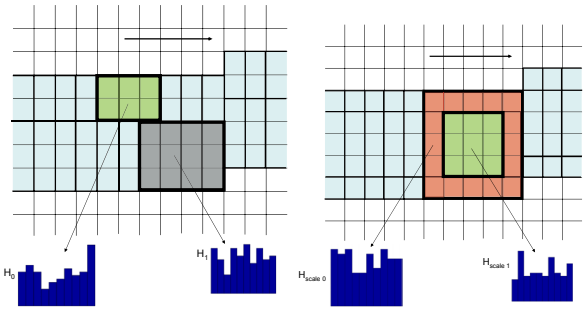


Fig. 4. Examples of a higher-order histogram-based feature (i.e., spatial arrangement of local histograms) constructed using the distributive histogram. Note that for these particular histograms, two histograms per column are kept track.

Analogously, the algorithm can be extended to higher dimensions while maintaining the $O(1)$ per pixel complexity in a straightforward manner. For example, for video analysis (i.e., three-dimensional space consisting of two spatial and a time dimension) the column histograms may be replaced with planar histograms extending into the temporal dimension.

3.1. Extensions

This section presents several extensions to the basic distributive histogram approach, such as handling multiple scales and spatial arrangements of histograms and the computation of regional statistics, such as mean [7], covariance (cf. [18]) and higher order moments.

3.1.1 Non-rectangular windows and multiple scales

A drawback of using a single local histogram to represent objects is that topological relationships between features are lost. The use of multiple (possibly multi-scale) spatial arrangements of local histograms re-introduces a degree of topological information. A simple and effective way to accomplish this is to utilize the histogram distributivity property again, where rather than tracking a single histogram per column, we now keep track of multiple histograms per column which are combined together to realize the final histogram (see Fig. 4). This basic scheme can also be adapted to keep track of histograms at multiple scales. Thus, multiple target scales can be explored efficiently in a single pass of the image.

3.1.2 Regional statistics

The ability to rapidly compute regional statistics has proven to be a useful tool for a variety of computer vision and image processing applications, such as, matching, recognition, filtering, etc. The computation of the arithmetic mean is a

widely used technique realized by sliding window summation and extensively used in *e.g*. stereo matching [20]. Several authors have explored the exhaustive computation of the covariance matrix, *i.e*. second order moment. For example, [18] adapted the integral image technique [7] to compute covariance matrices of regional multi-dimensional data, while [19] employed integral images to compute higher order moments to approximate the image patch with polynomials for fast template matching. While both these instantiations demonstrated great speed, the overflow concern (hence, applicability to larger images) is even more serious as the integral images operate with the squared, cubed, *etc*. sums of values.

The central ideas behind the distributive histogram can also be adapted for computing regional statistics. Particularly, in case of the mean, $\mu = \frac{1}{N}\sum \mathbf{X}_i$, where $\mathbf{X}_i \in \mathbb{R}^n$ is the data at point i and N is the number of points in the region, the distributive histogram can be seen as a simple generalization of the sliding window technique for box average computation. Instead of computing local histograms by unionizing appropriate column histograms, we just operate on the sum of the original values; the average is trivially obtained by unionizing, *i.e*. adding appropriate column sums.

The covariance matrix computation is very similar to the first moment computation. Now, we compute $\mathbf{C} = \frac{1}{N}\sum \mathbf{X}_i\mathbf{X}_i^\top - \mu\mu^\top$, where μ is the mean as above. The sum of outer-products is computed in the same way as the sum of corresponding column outer product terms. Extension to higher-order moments is a straight-forward application of the approach above. Finally note that this approach avoids the overflow issue that plagues the integral image-based method.

3.2. Optimizations

Utilizing *Single Instruction Multiple Data* (SIMD) instructions available on current CPUs (e.g., MMX/SSE on the Intel platform and AltiVec on the PowerPC) significant computational savings may be realized. For instance, using SIMD instructions the arithmetic operations used to update the column and kernel histograms can be computed across multiple bins in parallel (i.e., vectorized operations). Significant further gains using SIMD instructions are expected in the near future as CPU makers enhance their SIMD feature sets.

A further optimization is possible by restricting attention to bin sizes that are powers of two. The expensive division can be eliminated by a relatively inexpensive bitwise shift operation. For example, for a 16-bin histogram the bin of a pixel from an image with integer values ranging between 0 and 255 can be determined by four leftward bitshifts (i.e., features coded with the higher 4-bits).

4. Analysis

4.1. Computational analysis

In the following section we provide a computational analysis for histogram matching using 2D images at a single scale. The entities we operate on are: $B \equiv$ number of bins, $r \equiv$ diameter of a target, $N \equiv$ size of image (height/width). The operations we consider are: $a \equiv$ addition/subtraction, $d \equiv$ division , $f \equiv$ floor and type conversion, $b \equiv$ bitwise shift.

The conventional method relies on the brute-force construction of the histogram for each spatial location. In particular, for each point in the target of diameter r it requires:

1. Find the histogram bin for a pixel: 1 division and 1 floor
2. Increment the aforementioned bin: 1 addition

The total computational cost of the conventional method is:

$$N^2r^2(d + f + a). \tag{2}$$

Huang's approach reuses the histogram from the previous location by subtracting out the data corresponding to the column left of the histogram and adding data from the column right of the histogram. More precisely the operations are:

1. Find the bin of the pixels of the column histogram to the right of the kernel histogram: 1 division and 1 floor each
2. Increment aforementioned bins: 1 addition each
3. Find the bin of the pixels of the column histogram to the left of the kernel histogram: 1 division and 1 floor each
4. Decrement aforementioned bins: 1 subtraction each

Since a column histogram is composed from r pixels, the above operations are performed r times. Also, the initialization of the kernel histogram is done once per row in the conventional way, which makes the total cost for Huang's method:

$$Nr^2(d+f+a)+2N^2r(d+f+a) = N^2r(d+f+a)(2+\frac{r}{N}). \tag{3}$$

The integral histogram approach consists of an initialization stage that constructs integral images for each histogram bin and an extraction stage. Initialization is a straightforward procedure that conceptually consists of two steps:

1. Determine which bin the current pixel value falls in: 1 division and 1 floor
2. Update the corresponding integral image location as in [7, 22] for each bin image, *i.e*. add value from the left and from the top of the pixel: $2a$ per image

Histogram realization consists of the extraction of regional data from each bin integral image:

1. For each bin extract regional sum of values from the corresponding integral images: 3 additions

The detailed description of this procedure can be found in [17]. Thus, the total cost of the integral histogram is:

$$N^2(B(d + f + 2a) + 3aB) = N^2B(d + f + 5a). \tag{4}$$

Finally, we calculate the computational cost of the proposed distributive histogram method, which consists of initialization and extraction. Initialization involves the one-time construction of the initial column histograms and involves the following steps:

1. For each pixel find its histogram bin: 1 division and 1 floor per pixel

Method	General cost	Power-of-2 bins cost
Conventional	$N^2r^2(d+f+a)$	$N^2r^2(b+a)$
Huang	$N^2r(d+f+a)(2+\frac{r}{N})$	$N^2r(b+a)(2+\frac{r}{N})$
Integral	$N^2B(d+f+5a)$	$N^2B(b+5a)$
Distributive	$N^2\left(2(d+f)+2(B+1)a+\frac{r}{N}(d+f+a)\right)$	$N^2\left(2b+2(B+1)a+\frac{r}{N}(b+a)\right)$

Table 1. Computational costs for all histogram-based search methods. $N \equiv$ image width/height, $B \equiv$ number of histogram bins, $r \equiv$ target diameter, a, d, f, b are the costs of addition/subtraction, division, floor and bit-wise shift operations, resp.

2. Increment the aforementioned pixel bin: 1 addition

The distributive histogram extraction stage consists of the following operations:

1. Find the bin of the "new pixel" of the column histogram to the right of the kernel histogram: 1 division and 1 floor
2. Increment the "new pixel" bin: 1 addition
3. Find the bin of the "old pixel" of the column histogram to the right of the kernel histogram: 1 division and 1 floor
4. Decrement the "old pixel" bin: 1 subtraction
5. Add the updated column histogram to the kernel histogram: 1 addition per histogram bin
6. Remove the old column histogram to the left of the kernel histogram: 1 subtraction per histogram bin

The total cost of the distributive histogram approach is:

$$Nr(d+f+a)+N^2(2(d+f+a)+2Ba)= \quad (5)$$
$$N^2\left(2(d+f)+2(B+1)a+\frac{r}{N}(d+f+a)\right).$$

Keep in mind that substantial lower computational costs can be realized due to the fact that the distributive histogram steps can be vectorized and run in parallel (see Sec. 3.2).

In addition to the above costs (2), (3), (4), (5), there is the overhead of computing the histogram similarity. However, these steps are identical for all four approaches and can be omitted from the analysis as they do not introduce relative differences in the costs.

The computational costs are summarized in Tab. 1. Since computational costs depend on many free parameters (image size, target size, number of histogram bins) and the cost of operations vary between architectures, it is hard to outline the best overall method in practice. Nevertheless, several conclusions can be drawn:

- Since r does not exceed N, Huang's method is always faster than the conventional approach because the former depends linearly on r rather than quadratically.
- The integral histogram should outperform Huang's method when the target diameter r is large and the number of bins B is relatively small. This situation is particularly common in practice, since histograms are more reliable when bins have adequate support.
- The initialization step for the distributive histogram is not significant when $r << N$ or $B >> \frac{r}{N}$ – this situation is very common in practice, since the search object typically occupies only a fraction of the image, *e.g.* $5-10\%$ of the area, which makes $\frac{r}{N}=0.05...0.1$. Considering this, the computation cost is similar to that of the integral histogram, but with smaller constants overall, which suggests that the distributive histogram should outperform the integral histogram.
- The distributive histogram is derived from Huang's method in a way to reuse previous computations more efficiently; thus, it is expected to be faster by design. According to the derived cost expression it is indeed the case, especially for large target diameter r and relatively small number of bins B – quite a natural configuration, as discussed above.
- Unlike the conventional and Huang's approaches, the computational costs of the integral histogram and distributive histogram are independent of the target size.

Method	Memory
Conventional	1
Huang	B
Integral	N^2B
Distributive	NB

Table 2. Memory requirements for each of the 4 approaches. $N \equiv$ image width/height, $B \equiv$ number of histogram bins.

4.1.1 Power-of-2 bin sizes

A substantial reduction in computation can be achieved by restricting the bin quantization to powers of 2. The steps for computing bin location for a value can be implemented via bitwise shift operations: Given integer data and bin size of 2^i, where $i \in \mathbb{N}^+$, bin location is calculated via i bit-shifts that is realized as one operation on the majority of the contemporary architectures. Thus, all bin location calculation costs, $d+f$, in Tab. 1 can be effectively substituted with a single b operation and is reflected in the third column of the table.

4.2. Memory analysis

The analysis of memory requirements reveals yet another important advantage of the distributive histograms. Without loss of generality, consider square images of size $N \times N$ and histogram with B bins. In general, any possible approach requires N^2 units of memory for the image and B units to specify the target and some bookkeeping memory that is independent of image and histogram sizes – we do not consider

those requirements in the following as they are uniform across all possible methods.

The conventional approach does not rely on any precomputed data; we can consider it to be "memoryless". Huang's approach relies on the histogram obtained in the previous iteration, which takes up only B units of explicit storage. The integral histogram precomputes an integral image for each bin, which results in a memory cost of N^2B units. Finally, the distributive histogram requires the storage for the column histograms, which is NB. The memory requirements of the respective methods are summarized in Tab. 2.

Generally, speed improvements are realized by employing more memory in order to reduce redundancy in computation. This trend is consistent with moving from the conventional to Huang's method to the integral histograms. In contrast, for the distributive histograms, the computational improvement is accompanied by huge memory savings in comparison with the integral histogram, which in practice results in better cache coherence and even further speed-up.

Moreover, the distributive histogram is not prone to overflow error that plagues the integral histogram approach. This in turn does not limit the image and histogram sizes that can be utilized with the distributive histogram. In fact, while the conventional, Huang's and distributive histogram methods can operate with *e.g*. 1-byte memory units (for the typical case of 8-bit images), the integral histogram require much larger bit-depth, *e.g*. 4-byte units, to avoid overflow even for moderate size images, which implies even greater storage burden for the integral histogram approach.

5. Evaluation

In order to establish the computational improvement in practice, all four histogram-based search methods were implemented in C. All experiments were conducted on a AMD Opteron 850 processor with 1 MB cache and 16 GB of RAM. We stress that the practical comparison of the four approaches is more revealing than a theoretical comparison alone, as memory, cache coherence, the nature of the arithmetic operations and vectorization influence the practical performance.

Figure 5 shows the computational times and memory loads for all four approaches considered in the paper relative to the conventional approach – the latter is the slowest, but takes no memory. Conclusions derived from the theoretical analysis of Sec. 4 are verified to be correct. In terms of performance, the integral histogram outperforms Huang's method for large image sizes and small number of bins, though at the significant expense of storage requirement. Interestingly, when the number of bins is large, Huang's method is faster than the integral histogram, although performance of the former does not scale well with increases to the target size. Finally, the distributive histogram exhibits the best performance overall and its memory requirements are noticeably lower than for the integral histogram.

In [17], the authors theoretically compare the conventional and integral histogram approaches and conclude that the latter is computationally superior. In the following, we concentrate on the comparison of the proposed distributive histogram method with the state-of-the-art integral histogram.

Figure 6 shows the computational improvement of the distributive histogram-based method over the integral histogram for exhaustive search over a range of image sizes. The target size was held fixed as the costs of both methods do not depend on it. As can be seen, the distributive method significantly outperforms the integral histogram, with greater improvements realized as the search image size or number of histogram bins increases. This improvement is apparent for two reasons. First, the distributive histogram search is computed in a single pass of the image (modulo initialization of columns), while the integral histogram requires two full passes. Secondly, the small memory footprint of the distributive-based method is highly cache friendly compared to the integral histogram-based method which requires the allocation of one integral image per bin. More importantly, the extensive memory requirements of the integral histogram preclude its use even for moderate size images with colour 3D tensor histograms (16^3 bins in our case), while distributive histograms can be applied unobtrusively in such cases.

In conclusion, our evaluation suggests that the distributive histogram realizes an average five-fold speedup and thousands of times less memory in comparison to the integral histogram, this is especially true when large image and histogram sizes are considered.

5.1. Object detection example

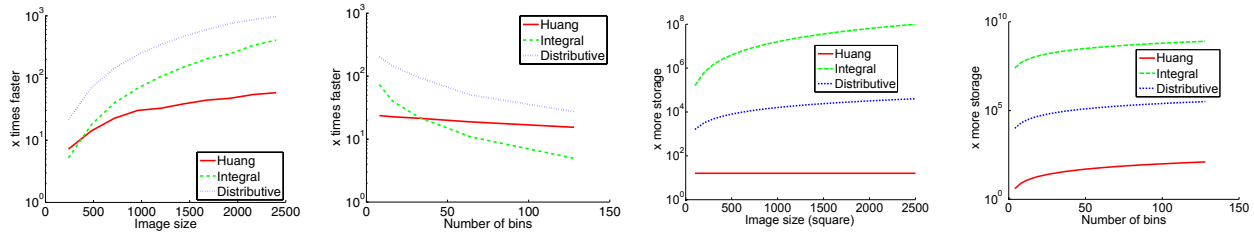
Figure 7 shows detection results for a traffic sign search example. In this example, a 16^3-bin (RGB) colour histogram was used. Although the distributive histogram and integral histogram methods compute identical similarity maps, the distributive histogram method runs in 868 msec and occupies 1.25 MB of memory, while the integral histogram method takes 5295 msec and occupies 1200 MB of memory.

For images of approximately 1 megapixel (1280×960), which is typical of contemporary digital cameras, the integral histogram requires 19200 MB of memory. This surpasses even our available RAM and requires access to much slower virtual memory. However, the distributive histogram requires only 5 MB of RAM for the same sized image.

Further speed improvements may be realized by utilizing the common speed up techniques outlined in Section 1.

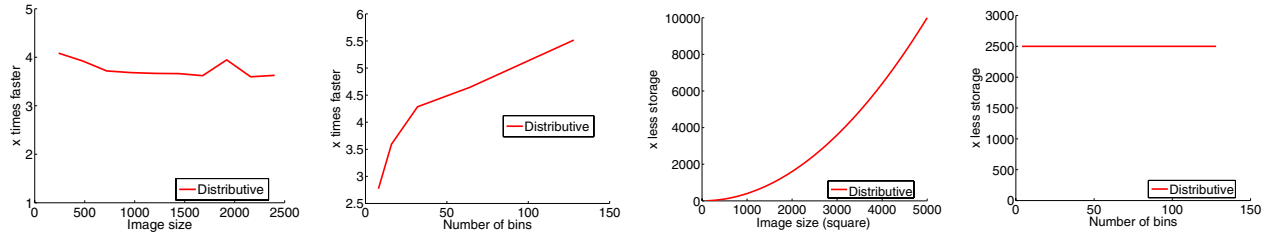
5.2. Tracking example

Figure 8 shows a tracking example. In this case, a 16-bin (hue) histogram was used. The distributive histogram method attains real-time performance taking only 2 msec (or 500 fps) and occupies 5 kB, while the integral histogram method takes 16 msec (or 62.5 fps) and occupies 4.8 MB, which results in eight-fold reduction in time and 960-fold reduction in space.



(a) (b) (c) (d)

Fig. 5. Computational improvement and storage requirements of Huang's method, integral and distributive histograms relative to the conventional approach. (a) Processing speed as a function of image size. Target size is 5% of the image size and 16-bin intensity histogram. (b) Processing speed as the function of number of bins. Image size is 960×720 and target size is 48×36. (c) Storage as the function of image size. (d) Storage as a function of number of bins. Note the logarithmic scale on the y-axis.



(a) (b) (c) (d)

Fig. 6. Computational and storage improvements of the distributive histogram approach over the integral histogram method. (a) Processing speed as a function of image size. Target size is 5% of the image size and 16-bin intensity histogram. (b) Processing speed as the function of number of bins. Image size is 960×720 and target size is 48×36. (c) Storage as a function of image size. (d) Storage as a function of number of bins.

(a) (b) (c)

Fig. 7. Traffic sign search example. a) 19×19 target template, b) 320×240 input search image with a bounding box overlayed on the matched target region and c) corresponding $L2$ similarity (log scale) image based on a 16^3-bin (RGB) tensor histogram (darker intensities indicative of greater similarity). The distributive histogram detects the target in 868 msec using 1.25 MB of memory, while the integral histogram takes 5295 msec using 1200 MB of memory.

Local tracking methods (e.g., mean-shift tracker [6]), although fast are prone to failure when significant frame-to-frame displacement of the target occurs. The distributive histogram tracker is fast, memory efficient and also robust to significant target displacement by virtue of the fact that it computes the similarity measure exhaustively across each frame.

6. Discussion

In this paper, we have presented and validated a novel and substantially faster method for exhaustive image search than those traditionally employed. We conclude with a discussion on the relative merits of our approach compared to the recently proposed state-of-the-art integral histogram [17, 14].

The key property of the integral histogram is that histograms of arbitrary scales can be computed in constant time (following initialization). Beyond the ability to efficiently explore multiple scales, this property enables the construction of "advanced" features, such as the spatial arrangement or the multi-scale fusion of local histograms. However, this degree of freedom is not without its issues: (i) the initialization computations scale with the full image resolution. (ii) the integral histogram requires a memory footprint of *image width* $\times$ *image height* $\times$ *#bins* (i.e., one integral image per bin). This places a heavy requirement on memory and in turn precludes its use on large images with a large number of bins, *e.g.* 16^3-bin colour-based search on 1024×960 images was impossible to perform in our experimental settings with 16 GB of RAM. (iii) the integral histogram suffers from overflow issues. This places strict boundaries on the search image and histogram sizes used for processing.

In contrast, the distributive histogram initialization exhibits low computational overhead and memory footprint (i.e., only the kernel histogram and column histograms must be maintained), while allowing multi-scale and "advanced" exploration of the data. Moreover, overflow is not an issue for the distributive histogram. Given the trend towards the availability of higher resolution imagery, the computational and

frame 60 frame 910 frame 3060

Fig. 8. Fish tank tracking example. Top row: selected frames from a $320 \times 240 \times 4000$ input image sequence with the bounding box of the tracked region overlayed. Bottom row: corresponding $L2$ similarity scores (log scale) based on a 16-bin histogram operating on the hue channel of the HSV colour representation constructed from a 19×19 target template (darker intensities indicative of greater similarity). The distributive histogram method detects the target in each frame in only 2 msec (or 500 frames per second) and occupies 5 kB of RAM.

memory advantages of the distributive histogram-based analysis are crucial.

In conclusion, we have performed a detailed comparison of all the major histogram construction methods for histogram-based search as well as presented a novel approach that significantly outperforms the current state-of-the-art in key practical situations.

Acknowledgements

The authors thank R. Wildes, J. Tsotsos and E. Leung for their valuable feedback. Also, the authors thank F. Porikli for clarifying details in [17]. M. Sizintsev is supported by NSERC PGS-D3 scholarship.

7. References

[1] D. Barnea and H. Silverman. A class of algorithms for fast digital image registration. *Trans. Comp.*, 21(2):179–186, 1972.

[2] S. Birchfield. Elliptical head tracking using intensity gradients and color histograms. In *CVPR*, pages 232–237, 1998.

[3] P. Burt. A pyramid-based front-end processor for dynamic vision applications. *Proc. IEEE*, 90(7):1188–1200, July 2002.

[4] P. Burt and E. Adelson. The Laplacian pyramid as a compact image code. *IEEE Trans. Comm.*, 31(4):532–540, April 1983.

[5] K. Cannons and R. P. Wildes. Spatiotemporal oriented energy features for visual tracking. In *ACCV*, pages 532–543, 2007.

[6] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *PAMI*, 25(5):564–577, May 2003.

[7] F. Crow. Summed-area tables for texture mapping. In *SIGGRAPH*, pages 207–212, 1984.

[8] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, pages I: 886–893, 2005.

[9] F. Ennesser and G. Medioni. Finding Waldo, or focus of attention using local color information. *PAMI*, 17(8):805–809, August 1995.

[10] J. Fung and S. Mann. Computer vision signal processing on graphics processing units. In *ICASSP*, pages 17–21, 2004.

[11] T. Huang, G. Yang, and G. Tang. A fast two-dimensional median filtering algorithm. *ICASSP*, 27(1):13–18, 1979.

[12] L. Kotoulas and I. Andreadis. Colour histogram content-based image retrieval and hardware implementation. *IEE Proc.-Circ. Dev. syst.*, 150(5):387–393, Oct. 2003.

[13] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *IJCV*, 43(1):29–44, June 2001.

[14] X. Liu and D. Wang. Image and texture segmentation using local spectral histograms. *Trans. IP*, 15(10):3066–3077, 2006.

[15] K. Nummiaro, E. Koller-Meier, and L. Van Gool. A color-based particle filter. In *GMBV*, pages 53–60, 2002.

[16] Perreault and Hebert. Median filtering in constant time. *Trans. IP*, 16(9):2389–2394, 2007.

[17] F. Porikli. Integral histogram: A fast way to extract histograms in Cartesian spaces. In *CVPR*, pages I: 829–836, 2005.

[18] F. Porikli and O. Tuzel. Fast construction of covariance matrices for arbitrary size image windows. In *ICIP*, pages 1581–1584, 2006.

[19] H. Schweitzer, J. W. Bell, and F. Wu. Very fast template matching. In *ECCV*, pages 358–372, 2002.

[20] C. Sun. Fast stereo matching using rectangular subregioning and 3D maximum-surface techniques. *IJCV*, 47:99–117, 2002.

[21] M. Swain and D. Ballard. Color indexing. *IJCV*, 7(1):11–32, 1991.

[22] P. Viola and M. Jones. Robust real-time face detection. *IJCV*, 57(2):137–154, May 2004.

[23] B. Weiss. Fast median and bilateral filtering. In *SIGGRAPH*, pages 519–526, 2006.