

A Near Optimal Acceptance-Rejection Algorithm for Exact Cross-Correlation Search

Anonymous ICCV submission

Paper ID 1186

Abstract

We describe a fast algorithm that searches for the k most likely locations of a template in an image according to the standard normalized correlations criterion. The algorithm is exact; it always finds the best matches. Its speed is achieved by utilizing an acceptance-rejection pruning scheme, applied to easily computed bounds on the normalized correlation values. The main idea is to use priority queues to decide the order in which the bounded matching candidates need to be considered. Previously proposed rejection schemes require the value of a rejection threshold that had to be provided or estimated from the data. Our algorithm does not use such thresholds explicitly, but performs as well as if the perfect rejection threshold is known. The performance of the algorithm with respect to an acceptance threshold is non-optimal, but we show this has little effect on the overall run time. Experimental results show that the algorithm requires roughly 15 arithmetic operations per pixel to search and detect the best match when an accurate match exists. When an accurate match does not exist the number of operations per pixel may be several hundred. Our experiments show that these results compare favorably with previously proposed exact matching schemes.

1. Introduction

Much of the recent work on fast template matching is based on the observation that many unlikely candidates can be pruned with little computation. This can be done in a probabilistic sense, rejecting candidates that are most likely not a good match [6], or in an exact sense, utilizing bounds on the match measure. Recent examples of the latter include [1, 3, 4, 5, 10].

All these exact methods utilize pruning using efficiently computed upper bounds. Let T_r be a threshold value with the property that at least one matching candidate has value of at least T_r . (For example, T_r can be the match value

computed exactly for one of the candidates.) Then any candidate with an upper-bound value below T_r cannot be the best match and can be pruned.

Pruning using upper bound values was shown to be very effective when accurate threshold values are available. In the studies described in [1, 4, 10] a good threshold is assumed to be known; in [3, 5, 10] it is computed incrementally as part of the algorithm. These studies have shown that the performance of the pruning approach depends heavily on the selected threshold. They proposed sophisticated and expensive methods for computing such thresholds. In fact, as reported in [3, 5], in many cases the computations related to threshold estimation dominate the overall performance of the algorithm.

Observe that the best possible rejection threshold is the value of the best match which is typically unknown. We describe an algorithm that behaves as well as if such threshold is known. By this we mean that the algorithm never evaluates the lower and upper bound values of a matching candidate if it would have been rejected by the optimal rejection threshold. This requires precise monitoring of the lower and the upper bounds of all candidates, which we implement in a binary heap. In our experiments the binary heap cost is non-negligible, but our experimental results show that the total complexity of our algorithm is still superior to the current state of the art.

The paper is organized as follows. Section 2 introduces some of the notation that we use throughout the paper. General bounds on normalized cross correlation that can be derived from orthogonal projections are discussed in Section 3. Our particular choice of orthogonal projections is the Walsh-Transform basis functions. The computational aspects of this choice are briefly discussed in Section 4. The algorithm that uses the bounds to perform the search is discussed in Section 5. Experimental results are shown in Section 6.

2. Some Notation

We denote the template to be detected by λ , and the candidates by y_1, \dots, y_m . The candidates correspond to all possible locations in an image, so that m is typically very large. Our goal is to compute the k best matches for λ among the m candidates. The match measure that we consider is the normalized cross correlation, which is standard in computer vision (see [7]). In vector notation it can be expressed as:

$$J(\lambda, y) = \frac{\lambda' y}{|\lambda| |y|} \quad (1)$$

Previous work on fast template matching using this measure include [5, 8]. Upper bounds on $J(\lambda, y)$ were derived in [5] for the special case of disjoint subsets. We generalize that result to arbitrary orthogonal projections, and use Walsh Transform kernels as the orthogonal projections. In particular, we show how to efficiently compute lower and upper bounds on $J(\lambda, y)$, with increasing degrees of accuracy. Let d denote the degree of accuracy, and let $l^d(y_i)$, $u^d(y_i)$ be the lower and the upper bounds on J computed at accuracy d . We assume that there is always a maximal degree of accuracy that we denote by N . At N the bounds are accurate. In summary, For all values of i and d the bounds satisfy:

$$\begin{aligned} l^d(y_i) &\leq l^{d+1}(y_i) \leq l^N(y_i) = J(\lambda, y_i) \\ &= u^N(y_i) \leq u^{d+1}(y_i) \leq u^d(y_i) \end{aligned}$$

3. Bounding Normalized Cross Correlations

The template λ and the candidates y_1, \dots, y_m are viewed as vectors in \mathbb{R}^n , where n is the number of pixels. Consider a partitioning of \mathbb{R}^n into the two orthogonal subspaces P and Q . Any vector $x \in \mathbb{R}^n$ can be expressed as a direct sum of its projections on P and Q :

$$x = x_p + x_q$$

We need the following form of Cauchy's inequality [9] for bounding the dot product of the two vectors λ, y :

$$-|\lambda| |y| \leq \lambda' y \leq |\lambda| |y|$$

Observe that

$$\lambda' y = \lambda'_p y_p + \lambda'_q y_q$$

Using the Cauchy inequality on $\lambda'_q y_q$ gives the following bounds on normalized correlations:

$$\frac{\lambda'_p y_p - |\lambda_q| |y_q|}{|\lambda| |y|} \leq J_N(\lambda, y) \leq \frac{\lambda'_p y_p + |\lambda_q| |y_q|}{|\lambda| |y|}$$

When the orthogonal subspace P is defined in terms of N orthogonal kernels v_1, \dots, v_N it is possible to compute

the bounds at increasing degrees of accuracy by the following simple iterative scheme. The bounds at accuracy $d-1$ are computed from the subspace defined by $V_{d-1} = (v_1, \dots, v_{d-1})$. The bounds at accuracy d are computed from the subspace defined by $V_d = (v_1, \dots, v_d)$. Given the bounds computed at degree $d-1$ and v_d compute:

$$f = v'_d \lambda, \quad g = v'_d y \quad (2)$$

Then apply the following update formulas:

$$\begin{aligned} \lambda'_p y_p &\leftarrow \lambda'_p y_p + fg \\ |\lambda_q|^2 &\leftarrow |\lambda_q|^2 - f^2, \quad |y_q|^2 \leftarrow |y_q|^2 - g^2 \end{aligned} \quad (3)$$

The iteration is initialized with:

$$\lambda'_p y_p = 0, \quad |\lambda_q|^2 = |\lambda|^2, \quad |y_q|^2 = |y|^2$$

4. Walsh-Transform Kernels

This section briefly describes the method used in our experiments to evaluate projections on Walsh kernels. For alternative methods see [1, 4]. Let w be a Walsh kernel. The projection of the image p on w can be expressed as follows:

$$\begin{aligned} p' w &= \sum_{i,j} p(i,j) w(i,j) = 2 \sum_{w(i,j)=1} p(i,j) - \sum_{i,j} p(i,j) \\ &= 2R_w - \alpha_{00} \end{aligned}$$

where α_{00} is the projection of p on the first Walsh kernel and R_w is the sum of the pixel values in p over the rectangles of "1" value in w . These sums can be computed using the "integral-images" approach [11]. Therefore, the complexity of this method is proportional to the number of corners in a Walsh kernel.

5. The Algorithm

This section describes our main result: an algorithm that uses lower and upper bounds to compute the k best matching candidates. (There are only minor differences between a version that detects the single best match and the one that detects the k best matches.) The algorithm gets as input a list G containing the set of all possible match locations that are to be considered. It starts by computing the initial bounds of degree 1 of accuracy for all candidates and then prunes G with the k^{th} smallest upper bound. Among the remaining candidates those with the k lowest lower-bound values are put in the "A" list, and all other candidates are put in the "B" list

$$\begin{aligned} A &= \text{the } k \text{ candidates with largest } u(y) \text{ in } G \\ B &= G \setminus A = \{y \in G; y \notin A\} \end{aligned} \quad (4)$$

Input: a list G of candidates.

1. Compute the initial bounds for all $y \in G$.
2. Set L to be the k^{th} largest lower bound in G . Remove from G all y satisfying $u(y) < L$.
3. Compute the subsets A, B according to (4), and the two extreme candidates y_k, y_{k+1} according to (5).
4. **Iterate:**
 - 4.1 If $l(y_k) \geq u(y_{k+1})$ terminate with A .
 - 4.2 If $u(y_k) \geq u(y_{k+1})$:
 - 4.2.1 Increase the accuracy of y_k bounds.
 - 4.2.2 Compute a new y_k according to (5).
 - 4.3 If $u(y_{k+1}) > u(y_k)$:
 - 4.3.1 *Optional Step:*
Compute $L = \max(l(y_k), l(y_{k+1}))$, and remove candidates y satisfying: $u(y) < L$ from B .
 - 4.3.2 Swap y_k in A with y_{k+1} in B .
 - 4.3.3 Compute new y_k, y_{k+1} according to (5).

Figure 1. The algorithm

Once the lists A, B are initialized, the extreme candidates y_k, y_{k+1} are computed according to

$$\begin{aligned} y_k &= \arg \min_{y \in A} l(y) \\ y_{k+1} &= \arg \max_{y \in B} u(y) \end{aligned} \quad (5)$$

The following condition shows that the lists A, B are “separated” with A containing the k candidates with the best matching values:

$$l(y_k) \geq u(y_{k+1}) \quad (6)$$

We refer to this as the *early-termination condition*. If it is satisfied, the algorithm terminates with the candidates in A as the optimal solution. Otherwise, the bound accuracy of the extreme candidate y_k is increased, until the *early-termination condition* holds, or different extreme candidates must be selected. The algorithm steps are detailed in Fig. 1. A flowchart of the main iteration steps is shown in Fig. 2.

5.1. Correctness

The algorithm terminates at Step 4.1, whenever the *early-termination condition* holds. This guarantees that all candidates in A are better than any candidate in B .

To prove termination first observe that Step 4.2.1 is always reached with $l_k < u_k$, and thus its degree of accuracy is below N . This cannot happen more than mN times. Second observe that Step 4.3.2 decreases the value of u_{k+1} . Since there is only a finite number of possible upper bound values (at most mN for all candidates) this step cannot be executed infinitely many times.

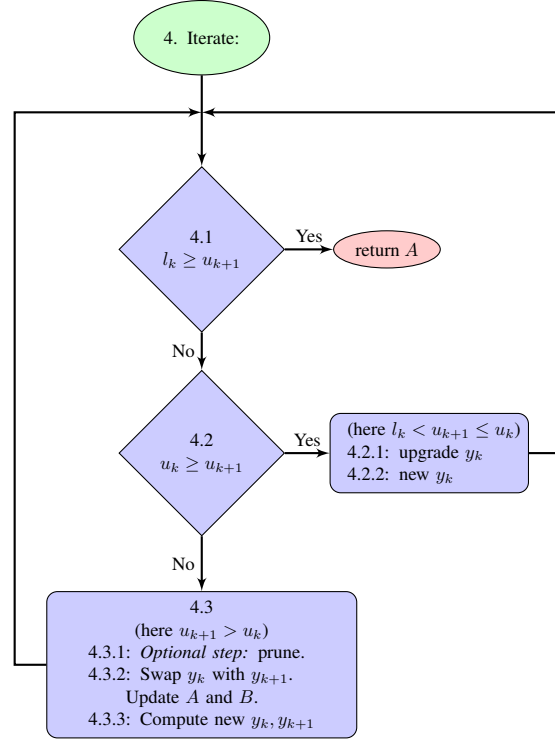


Figure 2. Flowchart of the main iteration steps

5.2. Complexity

In this section we analyze the run-time complexity of the algorithm. We show that the number of estimates of lower and upper bounds required by this algorithm is nearly the best possible.

The run time complexity consists of two components. The first is the computation of lower and upper bounds, carried out in steps 1 and 4.2.1. We refer to it as the *inherent complexity* of the algorithm. The second component is the creation and maintenance of the lists A, B , including the repeated computation of the extreme candidates y_k and y_{k+1} . We show that this can be done with priority-queues, and refer to this component as the *priority-queue complexity* of the algorithm.

5.2.1 Inherent complexity

As mentioned in Section 1 the inherent complexity of previously proposed algorithms depends on the accuracy of a pruning threshold, corresponding to the match value of the k^{th} candidate. We show that the algorithm performs nearly as well as algorithms that have access to a perfect threshold. Suppose we know the following two threshold values:

$$\begin{aligned} T_r &= \text{The } k^{\text{th}} \text{ largest match value} \\ T_a &= \text{The } (k+1)^{\text{th}} \text{ largest match value} \end{aligned} \quad (7)$$

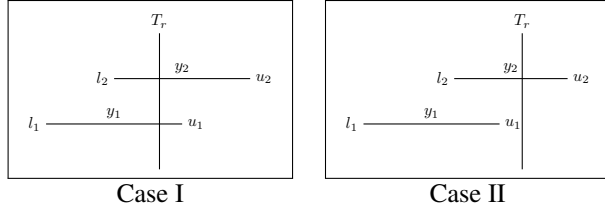


Figure 3. An illustration for Theorem 1. Here $k = 2$, $A = \{y_1, y_2\}$, and the extreme candidate in A is y_1 . In Case I the accuracy of y_1 bounds is increased. This cannot lead to Case II, where u_1 is to the left of T_r and y_1 can be pruned. T_r must be on the 2nd best candidate, but there is only one upper bound to its right. (Upper bound values of candidates in B are all to the left of u_1 .)

Then it is possible to apply the following *optimal* acceptance/rejection scheme:

$$\begin{aligned} \text{if } l(y) > T_a & \text{ accept } y \text{ as part of the solution} \\ \text{if } u(y) < T_r & \text{ reject (prune) } y \end{aligned} \quad (8)$$

Therefore, the accuracy of the bounds on y need not be increased if one of the conditions in (8) holds, but it must be increased if none of them hold.

Theorem 1. *The algorithm does not increase the bound accuracy for a candidate that can be pruned with a perfect threshold.*

Proof. We need to show that whenever the bound accuracy of y is increased then $u(y) \geq T_r$. The accuracy of y bounds is increased in Step 4.2.1, where y is in A . Therefore, there can be at most $k - 1$ candidates (all in A) with upper-bound values strictly above $u(y)$. These cannot include the candidate with the match value of T_r , since that one is the k^{th} largest candidate, with at least $k - 1$ candidates having upper-bound values above it. \square

See Fig. 3 for an illustration of the key observation. As shown in Fig. 4 the algorithm is not optimal with respect to the threshold T_a . Candidates that end up as part of the optimal solution may be evaluated to higher accuracy than necessary (if T_a is known). This applies to at most k candidates.

5.2.2 Complexity analysis

In evaluating the inherent complexity we ignore terms that are independent of the candidates and can be precomputed. Let \hat{c} be the fixed cost per pixel of the preliminary computations. In our implementation this consists of the evaluation of two integral images; one for computing sums of squared pixel values (for computing norms) and the other for computing pixel sums, that we use for computing projections on the Walsh kernels. For the other terms define:

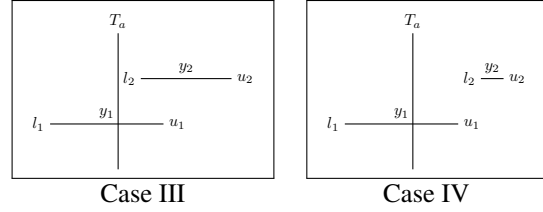


Figure 4. Here $k = 1$, and $A = \{y_2\}$. In Case III, if the value of T_a is known, the algorithm can terminate with y_2 as the best candidate. The algorithm will not terminate, and will keep improving the bounds of y_2 until $l_2 \geq u_1$, as shown in Case IV.

p_d - fraction of candidates evaluated at degree d .
 c_d - cost of updating bounds from degree $d - 1$ to d .

This gives the following formula for the inherent complexity:

$$m \sum_{d=1}^N c_d p_d = m \bar{c}_\alpha \sum_{d=1}^{N-1} p_d + e \bar{c}_\beta$$

Here e is the number of exact computations of the match measure, and \bar{c}_β is the cost of computing an exact match. The value of \bar{c}_α is the (weighted) average cost of updating a bound:

$$\bar{c}_\alpha = \frac{\sum_{d=1}^{N-1} c_d p_d}{\sum_{d=1}^{N-1} p_d}$$

The per-pixel inherent complexity is:

$$\text{Inherent complexity per-pixel} = \hat{c} + \bar{c}_\alpha \sum_{d=1}^N p_d + \bar{c}_\beta \frac{e}{m}$$

Let \bar{q} denote the per pixel cost of inserting a candidate into a queue, and let \hat{q} be the per pixel cost of initializing the priority queues. We have:

$$\text{Priority queue complexity per-pixel} = \hat{q} + \bar{q}$$

Combining the inherent complexity with the priority queue complexity the per-pixel complexity of the algorithm is given by:

$$\hat{q} + \hat{c} + \bar{q} + \bar{c}_\alpha \sum_{d=1}^N p_d + \bar{c}_\beta \frac{e}{m} \quad (9)$$

The value of $\sum_{d=1}^N p_d$ is bounded by N , but it is typically much smaller. It depends on the method in which the bounds are computed but not on other implementation details of the algorithm. In particular, using Walsh kernels we find that it grows very slowly as a function of N and can be treated as a constant.

5.3. An Implementation with Binary Heaps

We implemented the priority queues of both A and B as binary heaps [2]. The A heap was implemented as a MIN-HEAP, ranking the candidates in A according to their lower-bound values, and the B heap was implemented as a MAX-HEAP, ranking the candidates in B according to their upper-bound values.

In this implementation, retrieving the values of the extreme candidates y_k, y_{k+1} as the top elements of the two heaps is instantaneous. The two expensive operations are the deletion of candidates (at Step 4.3.1 of the algorithm) and the replacement of the candidate at the top of the heap (at Step 4.3.2). We did not implement deletion as in Step 4.3.1. (As noted, this does not affect the correctness of the algorithm.)

6. Experimental Results

This section describes experimental results with the algorithm. The input for each experiment consists of an image, a template, and the user selected parameters k, N . This determines the values of the following parameters:

- n - number of template pixels.
- m - initial number of candidates. This is slightly less than the number of image pixels since border locations where the template cannot fully fit are not considered viable candidates.
- k - desired number of matches.
- N - maximal degree of accuracy. At degree N the match value is computed exactly.

The following values are measured after termination:

- q_0 - number of swaps used for creating the heaps.
- q_1 - number of swaps used for updating the heaps.
- p_d - fraction of candidates evaluated at accuracy d .
- e - number of match measures computed exactly

The value of c_d was taken as the number of corners of the d^{th} Walsh kernel (see Section 4). The cost of computing a match value exactly was taken as $\bar{c}_\beta = 3n$.

The cost per pixel of computing an integral image is 2 [11]. The cost per pixel of computing an integral image of squared values is 3. Therefore, the value of \hat{c} , the per-pixel cost of computing the two integral images, is $\hat{c} = 5$.

There are two “machine-dependent” parameters that affect the run-time:

- H_1 : Cost of an arithmetic operation.
- H_2 : Cost of swapping two array elements.

The values of \hat{c}, \bar{c} in (9) are proportional to H_1 , and the values of \hat{q}, \bar{q} are proportional to H_2 . We count the run time in terms of arithmetic operations per pixel, so that the

relevant term for comparing the two is the ratio:

$$r = \frac{H_2}{H_1} \quad (10)$$

We found the value of r to vary roughly in the range of 2 – 10, depending on the hardware used, the amount of memory available, and the size of the array allocated for the heaps. In our experimental evaluation we use $r = 5$. The terms that appear in (9) are computed as follows:

$$\begin{aligned} \hat{q} &= rq_0/m, & \bar{q} &= rq_1/m \\ P &= \sum_{d=1}^{N-1} p_d, & E &= \bar{c}_\beta \frac{e}{m} \\ \bar{c}_\alpha &= \sum_{d=1}^{N-1} p_d/P, & \hat{c} &= 5 \\ \bar{c}_\beta &= 3n \end{aligned}$$

With this notation we can express the total cost in terms of the following 6 components (see Eq. (9):

$$\text{Total complexity} = \hat{q} + \hat{c} + \bar{q} + \bar{c}_\alpha P + E$$

We describe results of several experiments. In the first experiment the varying parameter was the amount of noise added to the template. In Experiment 2 we measure the algorithm run time while changing the template size. Experiment 3 compares the algorithm to the algorithm described in [5]. Experiment 4 shows results with $k > 1$

Experiment 1

In this experiment we evaluate the performance of the algorithm as a function of the amount of noise added to the template. The experiment was performed on the following 13 images from the USC database: “airport”, “boat”, “clock”, “couple”, “drop”, “elaine”, “girl”, “jet”, “lenna”, “man”, “mandrill”, “sailboat”, and “tiffany”. The Harris corner detector was used to automatically select 5 templates from each image, giving a total of 65 runs of the algorithm. All images were 512×512 , and the templates 64×64 . The value of N was kept constant at $N = 100$ Walsh kernels in all runs. The various complexity components of the algorithm, averaged over these runs, are shown in Fig. 5.

While there are modest increases in the values of P and \bar{c} the largest increase comes from the priority queue complexity and the exact computation of matching values. The rate of growth of the total as a function of the noise level appears to be exponential, shown as a linear relation in the logarithmic plot. See Fig. 8 for the same total plotted against σ in a non-logarithmic plot.

6.1. Experiment 2

We evaluate the algorithm performance with the template size as the varying parameter.

To keep all other parameters unchanged, we ran the experiments on pairs of images and templates obtained by

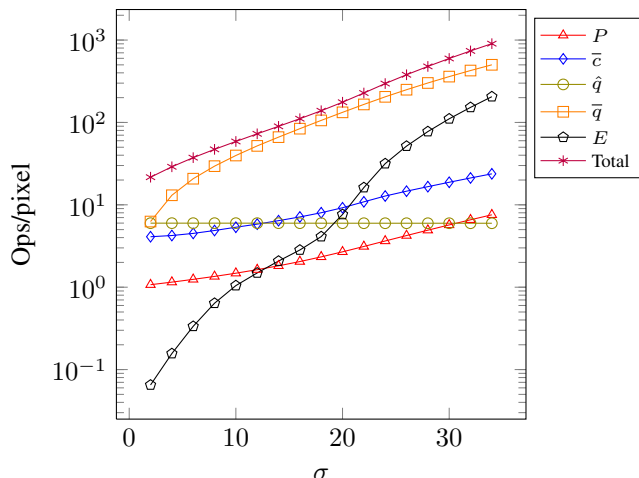


Figure 5. A logarithmic plot of the dual-bound algorithm costs, broken up into components, as a function of the template noise level.



Figure 6. The image and template in Experiment 2. The template is marked.

scaling both the template and the image by the same factor. (The image size itself is not relevant since the complexity components are measured per pixel.) It is evident from the plot in Fig. 7 that the template size has very little effect on the run time, In fact the only component which does increase noticeably is the queue complexity \bar{q} . The image and template are shown in Fig 6. The original image size is 1024×1024 , and the template size is 64×64 . A Gaussian noise of 0 mean with $\sigma = 10$ was added to the template. Both image and template were shrunk by factors of 1,2,4,8,16, and 32.

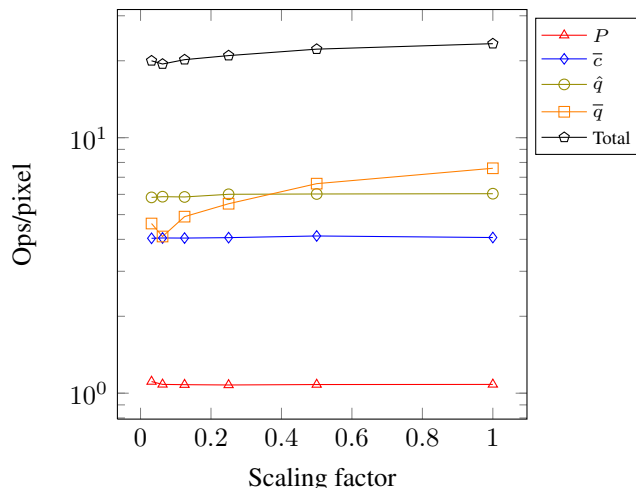


Figure 7. A logarithmic plot of the dual-bound algorithm costs, broken up into components, as a function of the scale of the image being examined. The image is shown in Fig. 6 with the test template location marked. The image size was reduced by factors of 1, 2, 4, 8, 16, and 32. Noise at a constant $\sigma = 5$ was added to the template after it was taken from the scaled image each time.

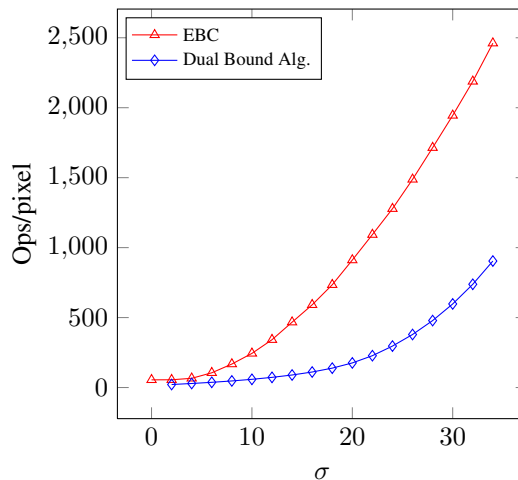


Figure 8. Comparison with the EBC algorithm

6.2. Experiment 3

We have implemented the Enhanced Bounded Correlation Algorithm (EBC) as described in [5]. This experiment was conducted with the same images and templates (a total of 65 runs) as in Experiment 1. The results are shown in Fig. 8. They show that the our algorithm outperforms the EBC, especially at high noise levels.

6.3. Experiment 4

Typically, setting $k > 1$ produces candidates surrounding the optimal match. In 9 we show the results of setting $k = 8$ in an image with repeated patterns. The match loca-



Figure 9. The algorithm applied with $k = 8$. The top image is the search image with the matching locations marked in black. The thicker lines on the first two cars are the result of multiple matches. The template is at the bottom left.

tions returned by the algorithm are outlined in black. Several locations overlap, producing the thick outline on the two vehicles to the right. The leftmost location was found only once with k at this setting.

7. Concluding Remarks

This paper describes a very fast algorithm for detecting image locations that match a given template according to the normalized correlations criterion. The main idea is to ignore image locations that can be proven non-optimal based on easily computed bounds and an implicit rejection threshold.

The idea of pruning appears in a lot of recent studies. Examples include [1, 3, 4, 5, 6, 10, 11]. In order for a pruning method to be effective one has to estimate a rejection threshold, a step that is typically considered as part of the overall algorithm. The unique property of our algorithm is that even though it does not use a rejection threshold explicitly, it performs as well as if the optimal rejection threshold is known. This comes with an overhead of manipulating priority queues, but our experimental results still show that the overall complexity compares favorably with the current state of the art.

Our estimate of r , the ratio of memory swap cost to arithmetic operation cost (see Eq. (10)) was $r = 5$. This value was measured on standard dual-core computers that come with highly optimized arithmetic logic units (ALU's) and floating point units (FPU's). We expect this ratio to be much smaller in implementations on simpler hardware such as the one used in hand-held devices. On such hardware we ex-

pect the performance of our algorithm to improve, when it is measured relative to the cost of arithmetic operations.

The technique of bounding match measures that was developed in Section 3 may be of independent interest. The algorithm itself is general, and can be used in any instance of pattern matching where it is possible to incrementally compute lower and upper bounds on matching values.

References

- [1] G. Ben-Artzi, H. Hel-Or, and Y. Hel-Or. The gray-code filter kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:382–393, 2007.
- [2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press and McGraw-Hill Book Company, 2001.
- [3] M. Gharavi-Alkhansari. A fast globally optimal algorithm for template matching using low-resolution pruning. *IEEE Transactions on Image Processing*, 10:526–533, 2001.
- [4] Y. Hel-Or and H. Hel-Or. Real-time pattern matching using projection kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:1430–1445, 2005.
- [5] S. Mattoccia, F. Tombari, and L. Di Stefano. Fast full-search equivalent template matching by enhanced bounded correlation. *IEEE Transactions on Image Processing*, 17:528–538, 2008.
- [6] O. Pele and M. Werman. Robust real-time pattern matching using bayesian sequential hypothesis testing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:1427–1443, 2008.
- [7] A. Rosenfeld and A. C. Kak. *Digital Picture Processing*. Academic Press, second edition, 1982.
- [8] H. Schweitzer, J. W. Bell, and F. Wu. Very fast template matching. In A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, editors, *Computer Vision - ECCV 2002*, number 2353 in Lecture Notes in Computer Science, pages 358–372. Springer-Verlag, 2002.
- [9] G. W. Stewart and J. Sun. *Matrix Perturbation Theory*. Academic Press, 1990.
- [10] F. Tombari, S. Mattoccia, and L. D. Stefano. Full search-equivalent pattern matching with incremental dissimilarity approximations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(1):129–141, January 2009.
- [11] P. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, May 2004.