Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Fixed Time Template Matching

Robert Finis Anderson, Haim Schweitzer
Department of Computer Science
University of Texas at Dallas
Richardson, TX USA

***Abstract*—The problem of finding a match for an image ('template') within a larger image is known as template matching. It is key to a variety of Computer Vision applications. Currently known template matching algorithms run in fixed time, or are guaranteed to find the best match. We present a novel algorithm which in many cases can guarantee that the best match is found. In other cases it finds a good approximation to the best match. This algorithm runs in fixed time (a.k.a. hard real time). It finds an optimal match very quickly when a good match exists.**

***Index Terms*—Machine Vision, Template Matching, Machine Vision Applications, Real Time**

I. Introduction

Template matching is a building block for many high level Computer Vision applications. Its runtime is often unfeasibly slow in raw form [1]. There has been much research into accelerating template matching for various applications. These methods can be viewed as occupying one of two groups. In one group, algorithms are capable of running in fixed time but are not guaranteed to find the best match according to the chosen error measure (for example [2], [3], [4], [5], [6], [7]). Recent research into template matching has produced a second group which guarantees finding the best match [8], [9], [10], [1], [11], [12]. We observe that the run-times of these algorithms are data-dependent.

In a hard real-time environment all tasks must be completed within a given time limit, and it is the responsibility of a scheduler to ensure that this happens [13]. If the amount of time required for a task is unpredictable, the system must be designed assuming that the task will run in worst-case time to guarantee that the task will complete before the deadline [13]. Therefore, the greater the worst-case run-time, the more CPU time will be wasted. Previous work in algorithms guaranteed to produce the best match did not take this into consideration. According to results shown in [8] the cost of matching a 64x64 template to a location in a 512x512 image varied by a factor of over 1000. The results of [10] show variations in run-time by a factor of 50, and [1] shows variations of 10 or more.

We present an algorithm which satisfies the goal of guaranteed run-time in template matching. It does this by taking advantage of available computing power to produce the best answer it can find within an allotted time. In many cases the algorithm finds the best possible match, and otherwise returns a close approximation. The algorithm does this by keeping an 'answer set' of current potential best matches. When the algorithm is stopped, it evaluates the current 'answer set' and returns the best match from that set.

Other algorithms can be pre-tuned to run faster and less accurately, or slower and more accurately (e.g. [14]), and it is always possible to subsample the search space with a corresponding loss in detail. This is the first algorithm to progressively search for the best possible match while maintaining the capability of running in fixed-time.

This paper is organized as follows. Section II describes the workings of the algorithm, including a brief proof of its correctness. Section III measures its performance in average cases for different situations. Lastly, we present concluding remarks and directions for future developments.

II. The Early Termination Algorithm

Throughout this paper we make use of the l_2 norm based distance measure (i.e. the Euclidean distance) between template and image subwindow. We denote the l_2 norm of a vector x by $|x|$.

Let the template to be detected be represented by a vector $\lambda \in \Re^n$. We consider each subwindow y_i of the search image I a potential match. I contains m pixels. The subwindows may overlap, and all contain n pixels. For convenience we define $Y = \{y_1, y_2, \ldots y_m\}$ to be the set of all potential matches. The error for the i^{th} candidate (or sub-window) is:

$$E_i = |\lambda - y_i|^2$$

The purpose of template matching is to attempt to find the y_i which minimizes E_i. The algorithm we propose is based on ideas described in [15]. In the next section, we briefly review the idea of using two bounds to accelerate template matching [15].

A. Bounds

Let the projection of λ and y_i onto orthogonal subspaces W and C be y_i^w, λ^w and y_i^c, λ^c. Applying the triangle inequality

$$| \, x \, | - | \, z \, | \leq | \, x - z \, | \leq | \, x \, | + | \, z \, |$$

to the rightmost term in

$$|\lambda - y| = |\lambda^w - y^w| + |\lambda^c - y^c|$$

we get:

$$|\lambda^w - y_i^w|^2 + (|\lambda^c| - |y_i^c|)^2 \leq |\lambda - y_i|^2 \quad (1a)$$

$$|\lambda - y_i|^2 \leq |\lambda^w - y_i^w|^2 + (|\lambda^c| + |y_i^c|)^2 \quad (1b)$$

This represents upper and lower bounds on the error E_i. Note also that since $x^w \cdot x^c = 0$ for arbitrary x, we also have

$$|\lambda^c|^2 = |\lambda|^2 - |\lambda^w|^2 \quad (2)$$

In our implementation W actually represents a set of mutually orthogonal vectors (which we will call kernels in keeping with other work in this field, i.e [10]) which is in turn a subset of an arbitrarily ordered set of mutually orthogonal kernels W'. We use W to represent the first d vectors from the set W'. During the course of the algorithm, each y_i is progressively projected onto more and more of the set W', with C representing orthogonal complement of W (that is $C = W^{\perp}$). Furthermore, we define $l_d(y_i) = |\lambda^w - y_i^w| + (|\lambda^c| - |y_i^c|)$ and $u_d(y_i) = |\lambda^w - y_i^w| + (|\lambda^c| + |y_i^c|)$ as the lower and upper bounds after the projection of y and λ on the first d kernels. When we write $l(y_i)$ without an explicit value for d, we mean the highest d currently evaluated for y_i.

For our algorithm to work efficiently, it should be possible to quickly compute the projection of the image on the kernels. In our implementation, we make use of the Walsh Projection Kernels [9], [12], [10]. We order the Walsh Kernels by 'sequency' [16], a concept related to visual frequency in 2 dimensions.

B. Initialization

At initialization the algorithm requires two parameters: k (the number of members allowed in the set A, defined later) and d_0 (the initial number of orthogonal kernels onto which all subwindows y_i are projected). These values are not strongly data dependent, and d_0 can effectively be set to 0 for all purposes. Experimental results show that the algorithm performs well with k in the range of 3 to 50 for our data, which should extend to other applications. See section III for more details on k.

The algorithm is initialized by computing the Walsh projections up to kernel number d_0 for all y_i, then computing the corresponding $l_{d_0}(y_i)$ and $u_{d_0}(y_i)$ for each. Then algorithm then separates the data into two sets $A, B \subset Y$. If y_k is the y_i with the k^{th} smallest value of $l(y)$, then

$$A = \{y_i \text{ s.t. } l(y_i) \leq l(y_k)\} \tag{3}$$

$$B = Y \setminus A \tag{4}$$

Thus, $|A| = k$. If there exists a $l(y_i) = l(y_k)$ and there are already k elements in A, one is arbitrarily chosen and placed in B. A represents our current best guess at the optimal answer set. These steps are represented in lines 1-3 in fig 1. The algorithm maintains these properties at all times throughout its operation.

C. Iteration

The Early Termination Algorithm makes use of both the upper and lower bounds in eq. 1 to quickly estimate the y_i with minimum E_i. At each iteration the algorithm computes one more projection for one candidate, checks a condition, and potentially updates A and B. Since all the components of W' are mutually orthogonal, if we have y_i^w for the first d vectors in W', then y_i^W for the first $d+1$ vectors is equivalent to y_i^w plus the projection of y_i on the $d+1^{\text{st}}$ vector. Using this fact along with eq. 2 we are able to quickly calculate $l_{d+1}(y_i)$ and

EARLY TERMINATION ALG.(λ, Y, k, d_0)

```
1  CalcInitialKernels(Y, d_0)
2  InitA(Y, k)
3  InitB(Y, A)
4  while ! earlyTerm
       do
5          Find y_u, y_{k+1}
6          if u(y_u) ≤ l(y_{k+1})
7             then return y_exact = arg min_{y_i} E(y_i),
                   y_i ∈ A
8             else if l(y_u) ≤ l(y_{k+1})
9                     then compute l(y_u)_{n+1}
                            and compute u(y_u)_{n+1}
10                    else Swap(y_u, y_{k+1})

11 return arg min_{y_i} e(y_i), y_i ∈ A
```

Fig. 1. The Dual Bound Early Termination Algorithm

$u_{d+1}(y_i)$ using $l_d(y_i)$ and $u_d(y_i)$, respectively, along with the projection of y_i on the $d+1^{\text{st}}$ vector in W'.

In the pseudo code in fig. 1, $y_u = \arg\max_{y_i} u(y_i), y_i \in A$, and $y_{k+1} = \arg\min_{y_i} l(y_i), y_i \in B$. They thus represent the candidate with the largest upper bound in A, and the candidate with the lowest lower bound in B accordingly. The $\text{Swap}(y_u, y_{k+1})$ function removes y_u from A and y_{k+1} from B, then places y_u in B and y_{k+1} in A. Steps 5 through 10 continue until the condition on line 6 is satisfied, or until the algorithm receives a signal to early terminate. Since only one answer is required, we return only the match in A with the lowest E_i on line 7.

D. Analysis and Correctness

Conceptually the algorithm is trying to separate the sets A and B; as soon as it is shows that $\forall y_i \in A, \forall y_j \in B, E(y_i) \leq E(y_j)$, it terminates automatically.

Theorem: The algorithm is guaranteed to find the candidate with the lowest match value if allowed to run to completion.

Proof: (Sketch) At the end of each iteration in the algorithm, there are guaranteed to be k candidates in A. Additionally, $\forall y_i \in A, E(y_i) \leq u(y_u)$; thus we are always guaranteed k matches in A with an error below $u(y_u)$. Therefore, if the condition on line 6 in fig 1 is satisfied, and $u(y_u) \leq l(y_{k+1})$, then it must be true that $\forall y_i \in A, E(y_i) \leq u(y_u) \leq l(y_k + 1)$. Thus we have k candidates guaranteed to have a match value less than or equal to the value of the $k+1^{\text{st}}$ smallest lower bound. Suppose $\exists y_i \in B, y_j \in A \mid E(y_i) < E(y_j)$ and the above condition holds. This would mean that $l(y_i) < l(y_{k+1})$. Therefore y_i is guaranteed to be in A by the condition in eq. 3. Since $A \cap B = \emptyset$, this is clearly impossible since y_i is already in B[1]. The procedure on line 7 guarantees that the algorithm

[1]It may be true that $\exists y_i \in B, y_j in A \mid E(y_i) = E(y_j)$. This cannot be avoided, as there may be multiple candidates with the same match value, though in practice this is quite rare.

returns the smallest among these k. ■

In terms of memory the algorithm stores at most a single copy of each candidate, with the associated numerical bounds. While all the kernel projections of the template are typically pre-computed and stored, the projections of the various candidates (y_i) are evaluated once and the magnitude of their variation from the template is evaluated before the projection is discarded. Thus the overall memory complexity of the algorithm is $O(m)$ assuming that the template is much smaller than the image ($n \ll m$).

The worst case run-time is more difficult to analyze. The algorithm is guaranteed to terminate, given that one of the following two conditions holds:

a) $\exists d$ s.t. $l_d(y_i) = E(y_i) = u_d(y_i)$
b) We default to calculating $E(y_i)$ explicitly when d is greater than a certain threshold $\mathcal{D}$.

Although the first condition is true for Walsh Projections, we make use of the latter, to lower the computational cost, as even the best methods for calculating the Walsh Projections require at least 2 operations per pixel per kernel [10], and thus it is currently cheaper to default to direct computation at some point. We call this maximum number of projections $\mathcal{D}$. In practice very few candidates are ever computed explicitly.

Theorem: The algorithm is guaranteed to terminate.

Proof: (Sketch) At every iteration the algorithm must either return an answer (line 7), update the bounds of a given y_i to a higher d (line 9), swap a candidate between A and B (line 10), or go into early termination (line 11). First, note that there cannot be an unlimited number of updates — as noted previously, there is always a value for d for which the bounds are equal to the actual match value, and the candidate cannot be updated anymore. If we call this maximum value $\mathcal{D}$, then this step (line 9) can happen at most $O(m\mathcal{D})$ times. At this point, all candidates are fully evaluated, and it is trivial to find the k smallest and satisfy line 6. Next, note that the swap operation can only happen a limited number of times. The reason is that by the definition of A, on the first iteration, $l(y_u) < l(y_{k+1})$. This can only change if one of those bounds is recalculated at a higher d, and we have previously shown that is limited as well. This can lead to at most one swap, since we know that only one value has changed. Thus the swap function can also be called at most $O(m\mathcal{D})$ times. The other two options represent termination for the algorithm; therefore the algorithm is guaranteed to terminate after $O(m\mathcal{D})$ steps. ■

The average run time of the algorithm is examined experimentally in Sec III, and is shown to be much lower than the upper bound given above.

E. Early Termination

As noted above, the algorithm automatically terminates with a guaranteed optimal answer at line 7 if permitted to run long enough. Early termination operates by a different mechanism. The Early Termination Algorithm assumes it will be given a small amount of early warning before its time is up. Since in practice k is small ($\approx 3 - 50$), and we perform very little processing on those candidates, the advance warning time can be very short. The run-time of this step is not data-dependent, and will always be $\Theta(k)$.

When the algorithm receives the early termination message, it computes an approximate answer

$$y_{approx} = \arg\min_{y_i} e(y_i), y_i \in A \tag{5}$$

where $e(y_i)$ is the expected value of y_i, defined as the average of the lower and upper bounds on $E(y_i)$:

$$e(y_i) = \frac{u(y_i) + l(y_i)}{2} = |\lambda_a - y_a|^2 + |\lambda_b|^2 + |y_b|^2 \tag{6}$$

This is represented on line 11 of fig 1. In practice, we find the minimum of $2e(y_i)$ for $y_i \in A$. This equates to a single arithmetic operation per candidate in A since we already have $u(y_i)$, $l(y_i)$ $\forall y_i \in A$. Finding the minimum value in A consumes another arithmetic operation (plus one potential swap operation) per candidate. These steps can be combined for a total cost of at most $3k$ operations, independent of the input data. Since in most practical cases $k \ll m$ this cost is negligible. When the expected value $e(y_i)$ is in line with the actual value of $E(y_i)$, the minimum expected value will be the best match currently in A. Experimental evidence shows that this is typically very close to the optimal match (see Sec.III).

F. Implementation

Our implementation of the algorithm uses a heap data structure to represent the sets A and B described in section II, as it is a natural fit for finding the respective minimum and maximum of the two sets. Heap construction is $O(m)$, and removal of the min (or max) node, or insertion of a new node, is only $O(\log(m))$.

The Walsh projections are implemented using integral-image based methods [17]. A Walsh kernel w is composed of rectangular regions (each of which is uniformly positive or negative). The projection of the image p (of the same dimensions as w) 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 value "1" (white rectangles) in w. These sums are computed in three operations per rectangular region using integral images. The complexity of this method is therefore proportional to the number of rectangular regions in a given Walsh kernel. We have done studies comparing this straightforward method to the accelerated methods in [12], [10], and found that in the context of template matching, our method is considerably faster.

G. Cost Analysis

To clarify analysis, we break down the costs of the algorithm as follows. We count the average number of Walsh Kernels evaluated at each location in the image $P = \frac{1}{m}\sum_{\forall y_i} d(y_i)$. Additionally we compute the average number

of basic mathematical operations used to compute those kernels, $\bar{c}$. If $c(d)$ is the cost of computing the d^{th} kernel, then $\bar{c} = \frac{1}{P}\sum_{\forall y_i} d(y_i)c(d(y_i))$ We also compute the average cost of explicit computation without projection (i.e., computing at $\mathcal{D}$ as described in Sec. II-D). Define f as the number of candidates computed explicitly, and recall that there are n pixels per subwindow. Given that it takes 3 operations per pixel to calculate $E_i = \mid \lambda - y_i \mid^2$, we define this cost as $\bar{f} = \frac{1}{m}3nf$. Lastly we count the number of heap swaps required to construct and maintain A and B. We record the number of swaps during construction as

$$\hat{q} = \frac{5}{m}\text{swaps-construct}$$

The number of swaps used to maintain the heaps is

$$\bar{q} = \frac{5}{m}\text{swaps-maintain}$$

We multiply $\hat{q}$ and $\bar{q}$ by a constant (in our case, 5), because in our experiments the heap swap operation is correspondingly more expensive than the basic mathematical operations used in the other steps.[2] The sum of these values, plus the initialization cost of the integral images used to compute the Walsh kernel projections (a constant, 5 operations per pixel) is the total cost per candidate.

III. Experimental Results

To test the algorithm we made use of an experimental framework similar to that in [12]: for each image in our testing database (all of size 512x512), we evaluate the image using the Harris Edge Detector, and extract five random templates with high scores (strong corner features) of size 64x64. We then add zero mean Gaussian noise with σ varying from 5 to 75 to the templates and attempt to match them with their originating images. We then average the results over all templates and all images for each noise level. This experiment was repeated for various settings of k. For each of these settings, we early-terminate the algorithm after a predetermined number of operations per candidate location (per pixel, effectively) to simulate a real-time situation.

For the purposes of measuring the accuracy of the algorithm, we define y_{exact} to be the correct, best match in the image, and y_{approx} to be the answer the algorithm returns after early termination. We then measure $\bar{E} = E(y_{approx}) - E(y_{exact})$. This yields an idea of how similar the approximate match is to the best match without taking location into account. This also takes into account the fact that when noise levels are high, the best match and the next best matches are not as well differentiated as in the low noise case. If we consider a correct match to be one where less than $20^{\%}$ of the pixels differ between the approximate match and exact match by more than a threshold (as in [6]), and we set the threshold to 2 grayscale

[2]The cost of a heap swap is dependent on many factors, including the size of the processor cache, the size of the heap, and the clock speed and latency of main memory. Specialized hardware, or fast memory, can reduce this cost. On different pieces of typical PC hardware we have seen this cost vary from 3 to 15.

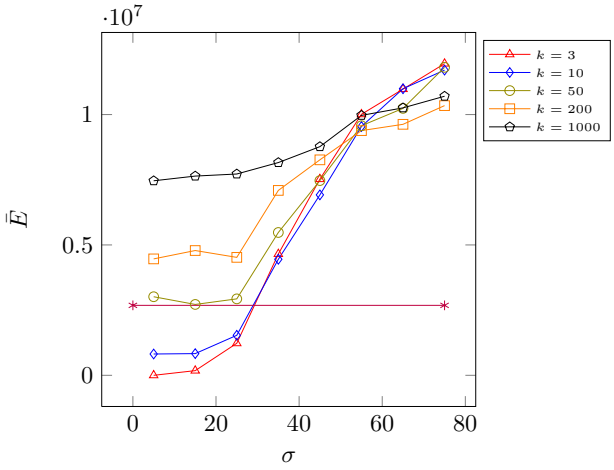

Fig. 2. Average difference between $E(y_{exact})$ and $E(y_{approx})$ after allowing the algorithm to run for 20 operations pixel. The horizontal line represents the line below which we consider it likely that the algorithm will find the optimal match. Noise varies from $\sigma = 5$ to 75.

levels, on average $\bar{E} < 2683044$ would be considered a correct match. This line is provided on the graphs for reference, and to allow comparison to approximate matching methods.

As can be seen in figs. 2, the algorithm performs quite well with small k with only 20 operations per pixel up to noise $\sigma = 25$. Beyond this level, the average difference between the approximate and exact match climbs steeply. The line at $2.6 * 10^7$ (explained above) provides a reference point. If the value of $\bar{E}$ is less than that, we consider it likely that there is substantial overlap between y_{approx} and y_{exact}; the lower $\bar{E}$ is, the more likely it is that the algorithm found the optimal match. This example shows higher performance with small values of k. In fact, the highest low-noise performance occurs at $k = 3$ here. For comparison, the state of the art exact method described in [11] requires 100-200 operations per pixel to find the correct answer at $\sigma = 20$.

When the algorithm is allowed to run to 30 operations per pixel (figs. 3) the situation is much better up to $\sigma = 35$, though not much improved above that. Again, small values of k show superior performance. For contrast, the method in [11] requires over 400 operations per pixel at these noise levels. Allowing 200 operations per pixel guarantees strong performance throughout the range of noise levels tested (fig. 4). Clearly, the accuracy of the algorithm drops with noise, however the performance remains strong when compared with other current methods.

Using a larger value for k would seem to yield a better chance that the optimal answer will be contained in A at the time of early termination. However, as k becomes larger, A becomes correspondingly more expensive to maintain, costing more operations per pixel (note fig. 5, where we can see the value of $\bar{q}$ rising while $\bar{c}$ and $\bar{f}$ drop). The values $\bar{c}$ and $\bar{f}$ not only indicate the amount of effort the algorithm expends in calculating the Walsh kernels and Euclidean distance, respec-

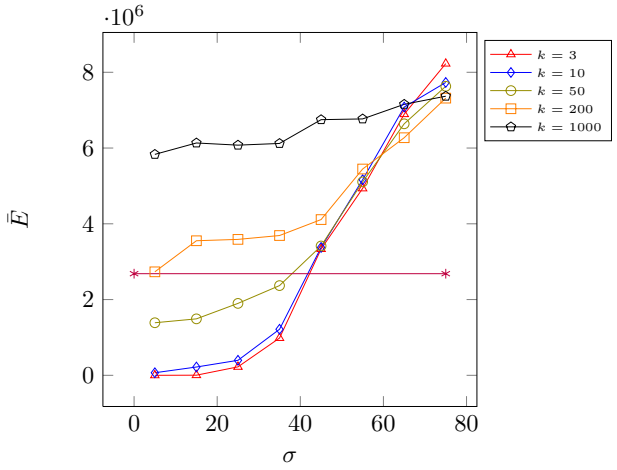

Fig. 3. Results of allowing the algorithm to run to 30 operations per pixel, with noise varying from $\sigma = 0$ to $\sigma = 75$

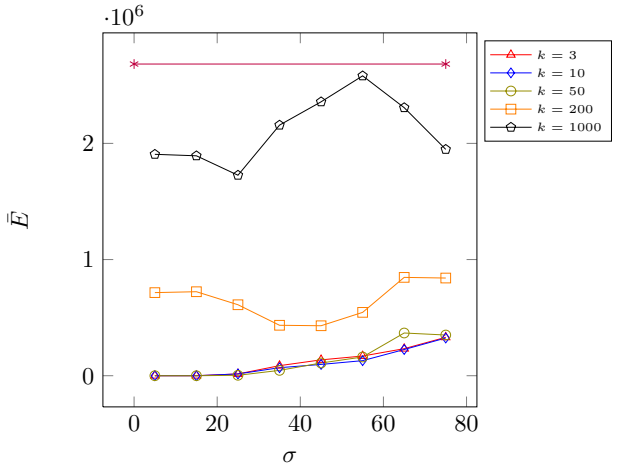

Fig. 4. Results from allowing the algorithm to run to 200 operations per pixel. The line denoting a likely optimal match is now well above the plots of $\bar{E}$ for all values of k.

tively, but they also directly denote the amount of information the algorithm has extracted from the image. This means that less of the algorithm's time is spent finding the best answer.

Additionally, as k increases, it becomes more and more difficult to separate the k^{th} value from the $k+1^{\text{st}}$. In fig. 6 we see that the difference between successive matches falls away quickly as k increases. Thus, when k is small, it is easiest to tell y_k from y_{k+1}, and the algorithm puts less effort into separating the two. When k is large, the values are very similar, and both must be evaluated to a higher value of d to differentiate them.

In fig. 7 I have run the algorithm on the classic image *lenna*, and early terminated the algorithm at varying times representing 15 to 30 operations per pixel with $k = 10$. Noise of $\sigma = 20$ has been added to the template. It can be seen that the algorithm initially finds a few poor matches for the image, and then rapidly focuses on a few neighbouring locations of the

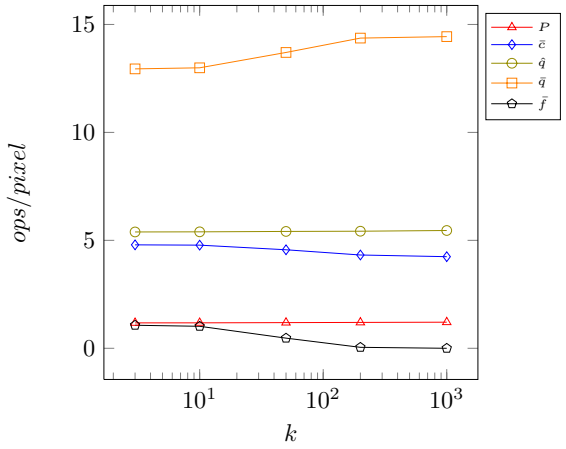

Fig. 5. Components of the cost of the algorithm, described in Sec. II-G, plotted against k varying from 3 to 1000. The algorithm was allowed to run the equivalent of 30 operations per pixel. It can be seen that as k increases, the cost of maintaining the queue rises, while the algorithm is forced to spend less time on computing the projections of Walsh kernels and exact distance.

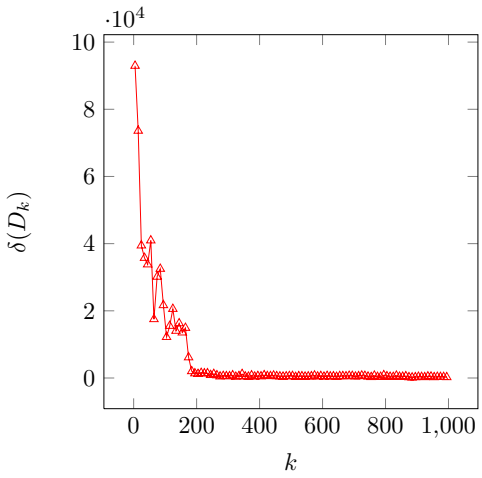

Fig. 6. If we order all $y_i \in Y$ by increasing value of $D(y_i)$, then this chart shows the value of $\delta(D_k) = D(y_{k+1}) - D(y_k)$. This effectively illustrates how tight the bounds on y_k must be to separate it from y_{k+1}. It can be seen that the higher k is, the lower $\delta(D_k)$ is, meaning the corresponding candidates are more difficult to separate.

match. Effectively, by 18 ops/pixel, the best match has already been found, and the algorithm is trying to prove that the match is the best possible. The reader is strongly encouraged to refer to a color copy for this image.

IV. Conclusion

In this paper we described a very efficient algorithm for template matching. The Early Termination Algorithm very quickly locates the best match for a template in the search image, in as little as 20 operations per pixel. Experiments showed the algorithm typically finds a match which is either the best match, or very close to it, within user-defined time-limits. We showed algorithm converges quickly on a set of

Fig. 7. Lenna, with match locations shown by '+' signs. The template is shown at lower right. It has had Gaussian noise with $\sigma = 20$ added to it. The varying colors represent the number of operations per pixel given to the algorithm before termination. The matches shown in red indicate matches given very little time (≈ 15 operations per pixel). By 18 operations per pixel, the algorithm has already settled on a location at or within a pixel or two of the best match. This run-time includes pre-processing, and is exceptionally fast when compared to the current state of the art. The reader is strongly encouraged to refer to a color copy.

answers which are all very close to the best match. The experiments also showed that the algorithm is comparatively robust to image distortions caused by noise. The algorithm has a small memory footprint of the same approximate size as the search image, and can run in situations where computational resources are limited. Our experimental results indicate that the run-time of the algorithm compares favorably with both fixed-time approximate methods and data-dependent exact methods.

Future Improvements

It may be possible for the algorithm to automatically early-terminate when it has consistently found the same best match over many iterations, which could dramatically lower the time to run to completion, especially at higher noise levels. Additionally, it would not be difficult to construct a variant of the algorithm designed to return a single answer guaranteed to be among the top k results, where k is a parameter supplied at instantiation. Lastly, the algorithm could be adapted to image database search, as it does not rely on the fact that the potential matches all come from the same image.

REFERENCES

[1] S. Mattoccia, F. Tombari, and L. D. Stefano, "Fast Full-Search equivalent template matching by enhanced bounded correlation," *Image Processing, IEEE Transactions on*, vol. 17, no. 4, pp. 528–538, 2008.

[2] A. Goshtasby, S. H. Gage, and J. F. Bartholic, "A Two-Stage cross correlation approach to template matching," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-6, no. 3, pp. 374–378, 1984.

[3] A. Rosenfeld and G. Vanderburg, "Coarse-Fine template matching," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 7, no. 2, pp. 104–107, 1977.

[4] H. Masnadi-Shirazi and N. Vasconcelos, "High detection-rate cascades for Real-Time object detection," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, 2007, pp. 1–6.

[5] W. Krattenthaler, K. Mayer, and M. Zeiller, "Point correlation: a reduced-cost template matching technique," in *Image Processing, 1994. Proceedings. ICIP-94., IEEE International Conference*, vol. 1, 1994, pp. 208–212 vol.1.

[6] O. Pele and M. Werman, "Robust Real-Time pattern matching using bayesian sequential hypothesis testing," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 30, no. 8, pp. 1427–1443, 2008.

[7] H. Schweitzer, J. Bell, and F. Wu, *Very Fast Template Matching*. Springer Berlin / Heidelberg, 2002, pp. 145–148. [Online].

[8] M. Gharavi-Alkhansari, "A fast globally optimal algorithm for template matching using low-resolution pruning," *Image Processing, IEEE Transactions on*, vol. 10, no. 4, pp. 526–533, 2001.

[9] ——, "A fast full-search equivalent algorithm using energy compacting transforms," in *Image Processing, 2001. Proceedings. 2001 International Conference on*, vol. 2, 2001, pp. 713–716 vol.2.

[10] G. Ben-Artzi and H. Hel-Or, "The Gray-Code filter kernels," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 3, pp. 382–393, 2007.

[11] F. Tombari, S. Mattoccia, and L. D. Stefano, "Full-Search-Equivalent pattern matching with incremental dissimilarity approximations," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 1, pp. 129–141, 2009.

[12] Y. Hel-Or, "Real-time pattern matching using projection kernels," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 9, pp. 1430–1445, 2005.

[13] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenstrm, "The worst-case execution-time problem; overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 1–53, 2008. [Online].

[14] J. Sochman and J. Matas, "WaldBoost - learning for time constrained sequential detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, 2005, pp. 150–156 vol. 2.

[15] H. Schweitzer, R. F. Anderson, and R. A. Deng, "A dual bound algorithm for very fast and exact Template-Matching," University of Texas at Dallas, Computer Science Dept., Technical Report UTDCS-02-09, Feb. 2009.

[16] E. W. Weisstein, "Walsh function – from wolfram MathWorld."

[17] P. Viola and M. J. Jones, "Robust Real-Time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, May 2004. [Online]. Available: http://dx.doi.org/10.1023/B:VISI.0000013087.49260.fb