Rotation invariant histogram filters for similarity and distance measures between digital images[1]

Kimmo Fredriksson
Department of Computer Science
University of Helsinki, FINLAND
Kimmo.Fredriksson@cs.Helsinki.FI

Abstract

Fast rotation invariant filtering algorithms for are presented for speeding–up the search of the position of the given pattern template from large image. The algorithms work for several distance/similarity measures. The algorithms give useful correlation information as such, and can be used as filtering algorithms for more sophisticated methods. The results are presented in the 2–D case, but extensions to 3–D are straight–forward. The algorithms are based on color histograms computed from the pattern. A technique is described for pruning the filtered positions incrementally, using simple, yet effective boundary that divides the search space in two.

1. Introduction

We present in this paper fast multidimensional filtering methods for finding possible positions and orientations of the given pattern from given search space.

Our approach to the problem is combinatorial. Examples of combinatorial pattern matching algorithms that work in two dimensions, but do not allow rotations are e.g. [3, 10, 13, 14, 15, 17, 18]. On the other hand, there are many non–combinatorial approaches to rotation invariant pattern matching, for a review, see e.g. [4, 20]. The only combinatorial methods, that come close to us in some respects, are [14, 17]. However, these do not address the pattern rotations. As stated in [2], a major open problem in two–dimensional (combinatorial) pattern matching is to find the occurrences of a two-dimensional pattern of size $m \times m$ in a two-dimensional text of size $n \times n$ when the pattern can appear in the text in rotated form. This was addressed from a combinatorial point of view first in [6], in which an online algorithm for exact search allowing pattern rotations was presented. Off-line searching (indexing) was considered in [5]. In [16] method called color indexing was introduced for application in image database indexing. These techniques use color histograms as indices of images stored in database. We also use the color histograms to produce fast filters for online algorithms.

Many traditional methods concentrate on constructing filters for recognition, that is, to compute a value somehow measuring the distance/similarity between two images of different size and orientation. Our goal is to efficiently search the given pattern template from large image, that is, to efficiently prune as many positions (and orientations) as possible from the search space at each single *signature* comparison, for many different similarity/distance measures. We do not consider how to evaluate the actual distance/similarity between the pattern template and the image. We concentrate on how to *quickly* decide where the distance/similarity is above/below some given *threshold* value. Often one requires that the occurrence of the pattern template in the image must be "good enough". Our thresholding method utilizes this fact to derive fast filtering algorithm. As the signatures to be compared, we use histograms of colors taken from certain circular areas of the template and the image.

Given two–dimensional image I, pattern template P, and a threshold value k, our filters work in times given in Table 1. In the given time bounds, the filter decides whether the pattern may have an occurrence in *any orientation*, for each location of I, for the distance/similarity measures stated. The first column is for our basic algorithms, and the second column for our improved algorithms. Most of the algorithms work for normalized distances also.

Our algorithms are very flexible, and can be adapted to many other distance/similarity measures also. Our closest competitor is the traditional correlation method using FFT, which takes time $\mathcal{O}(K|I|\log|I|)$, where K is the number of rotations sampled, and $|I|$ is the size of the input image. FFT needs also a lot of space, e.g. $16|I|$ bytes in typical implementations.

[1] A work supported by ComBi.

distance	time	time
Hamming	$\mathcal{O}(\|I\|r)$	$\mathcal{O}(\|I\|(r/s))$
Delta	$\mathcal{O}(\|I\|(r+\sigma))$	$\mathcal{O}(\|I\|(r/s+\sigma/s^2))$
SAD/SSD	$\mathcal{O}(\|I\|(r^2+\sigma))$	$\mathcal{O}(\|I\|((r^2+\sigma)/s^2+r/s))$
correlation	$\mathcal{O}(\|I\|(m+\sigma))$	-

Table 1. Summary of results. Here σ is the size of the alphabet, and $s \geq 1$ is the average *shift* our algorithms make, and $2r = \mathcal{O}(k^{1/2}) \leq \sqrt{|P|} = m$. SAD and SSD are sum of absolute differences of pixel values, and sum of squared differences, respectively. Delta was first defined in [7], see Sec. 4.2.

2. Problem definition

For simplicity, we give the algorithms in the two dimensionsional case. See [9] for a 3D version of the basic Hamming distance filter. All the other filters generalize also, in a straight–forward way.

Let $I = I[1..n, 1..n]$ and $P = P[1..m, 1..m]$ be two dimensional arrays of *point samples*, such that $m < n$. Each sample has a *color* in a finite ordered *alphabet* Σ. The size $|\Sigma|$ of Σ is denoted by σ. We call I the input *image*, and P the *pattern* template that is sought from I.

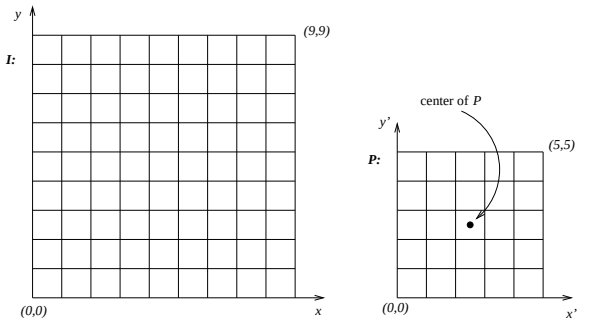

Figure 1. The array of pixels for I and P ($n = 9$, $m = 5$).

The arrays P and I are point samples of colors of some "natural" image. There are several possibilities to define a mapping from P to I, that is how to compare the colors of P to colors of I. Our approach to the problem is combinatorial. Assume that P has been put on top of I, in some arbitrary position. Then we will compare each color sample of P against the color of the closest sample of I. The distance between the samples is simply the Euclidean distance. This is also technically convenient. The Voronoi diagram for the samples is a regular array of unit squares, which we call *pixels*.

Hence we may define that the array $I = I[1..n, 1..n]$ consists of n^2 unit squares (pixels) in the real plane $\mathbf{R}^2$ (the (x, y)–plane). The corners of the pixel for $I[i, j]$ are $(i - 1, j - 1), (i, j - 1), (i - 1, j)$ and (i, j). Each pixel has a *center* which is the geometric center point of the pixel, i.e., the center of the pixel for $I[i, j]$ is $(i - \frac{1}{2}, j - \frac{1}{2})$. The array of pixels for pattern P is defined similarly. The *center* of the whole pattern P is the center of the cell in the middle of P. Precisely, assuming for simplicity that m is odd, the center of P is the center of cell $P[\frac{m+1}{2}, \frac{m+1}{2}]$. The colors are now associated with the center points of the pixels, that is, the centers are the sampling points.

Assume now that P has been moved on top of I using a rigid motion (translation and rotation), such that the center of P coincides exactly with the center of some pixel of I. The location of P with respect to I can be uniquely given as $((i - \frac{1}{2}, j - \frac{1}{2}), \theta)$, where $(i - \frac{1}{2}, j - \frac{1}{2})$ is the location of the center of P in I, and θ is the angle between the x–axis of I and the x–axis of P. The occurrence (or more generally, distance) between I and P at some location, is defined by comparing the colors of the pixels of I and P that overlap. We will use the centers of the cells of P for selecting the comparison points. That is, for the pattern at location $((i - \frac{1}{2}, j - \frac{1}{2}), \theta)$, we look which pixels of the image cover the centers of the pixels of the pattern, and compare the corresponding colors of those pixels. As the pattern rotates, the centers of the pixels of the pattern move from one pixel of I to another. In [6] it is shown that this happens $\mathcal{O}(m^3)$ times, so there are $\mathcal{O}(m^3)$ relevant orientations of P to be checked. The actual comparison result of two colors depends on the matching model.

More precisely, let us define the *matching function* M when P is at location $((u, v), \theta)$ as follows.

Definition 1 *For each pixel $P[r, s]$ of P, let $I[r', s']$ be the pixel of I such that the center of $P[r, s]$ belongs to the area covered by $I[r', s']$. Then $M(P[r, s]) = I[r', s']$.*

Hence the matching function is a function from the pixels of P to the pixels of I. We may assume that function M is uniquely defined; otherwise adjust θ "infinitesimally" such that no center of P hits the pixel boundaries of I. Note that M is many–to–one mapping. This fact must be reflected in our algorithms.

We use distance (similarity) function $d(p)$ to compute the distance between pixels p and $M(p)$. Our algorithms take the distance function as a "black box", and assume only that it can be computed in a unit time per pixel. Some other properties are also required. Our algorithms are based on Definition 1, but in Sec. 7 we discuss a particular generalization of it.

The distance between the pattern template and the image is defined as follows.

Definition 2 *The distance (similarity) D between I*

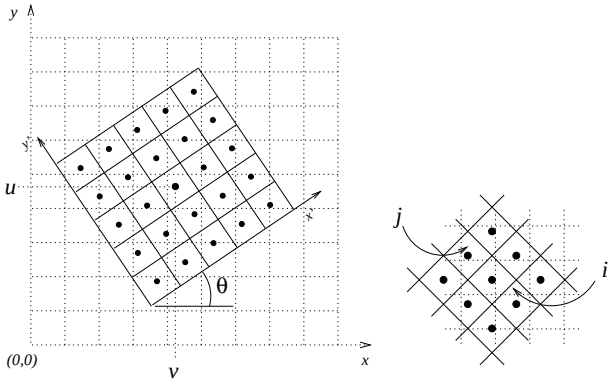

Figure 2. Illustration of M at location $((u, v), \theta)$. For each pixel of $P[r, s]$, the matching function $M(P[r, s])$ returns the pixel of I that covers the center of pixel $P[r, s]$. The matching functions is not one–to–one, for example the pixel i of I covers no center of P, and the pixel j of I covers two centers of P.

and P located at position $((u, v), \theta)$ in I is $D = D(P, I, ((u, v), \theta)) = \sum_{r,s} d(P[r, s])$.

In some cases we may want to normalize the distance (similarity). Some of our algorithms can do this also, without affecting the asymptotic time bounds. We consider the normalized distance D_n in Section 4.6. Typical distance functions $d(p)$ can be, e.g.,

- $d(p) = 0$, if $color(p) = color(M(p))$, and 1 otherwise
- $d(p) = |color(p) - color(M(p))|$,

where $color(p)$ denotes the color of pixel p.

In this work, we do not consider the problem of computing D or D_n. Instead, we concentrate the following *filtration* approach: given a parameter k, decide quickly if $D \leq k$ or $D_n \leq k$ (similarly $D \geq k$ or $D_n \geq k$), for any angle θ. In many applications one wants to find only reasonably good matches of the pattern template. Therefore, we may use some small threshold value k to discard uninteresting parts of I quickly, and thus solve the original problem efficiently.

Our algorithms have the following property. If they state that the distance $D \leq k$ (or $D \geq k$ for similarity), they may be wrong, so this must be verified using some other algorithm, but if they state that the distance $D > k$ ($D < k$ for similarity), its always correct. Hence the algorithms do not miss any good enough positions, that is, there are *no false negatives* in the filtration.

There is another way to look at our algorithms. Instead of filtering out unpromising positions of I, we may assume that our algorithms compute the approximate distance/similarity of P for each position of I. That is, we might define that P occurs in I at the position where the filter gives the highest peak of similarity. It is easy to find degenerate cases where this would give very bad results. However, despite of the flaws of this definition, in many applications this would give the desired result (see the experimental results).

3. Histogram composition

Our filter is somewhat similar to the 1D string matching filter by Grossi and Luccio [12]. Consider each pixel p of P such that the distance from the center of p to the center of P is at most $r \leq (m-1)/2$ (assuming odd m). These pixels fall inside a circle whose radius is r. Now calculate a *histogram* of the color distribution of those pixels. This histogram is (almost) rotation invariant, and together with corresponding histograms from I it can effectively filter unpromising positions of I. The histogram $\mathcal{H}$ has one bin $\mathcal{H}(c)$ for each color $c \in \Sigma$. $\mathcal{H}(c)$ tells how many pixels of color c there are in $\mathcal{H}$, and the size of the histogram is $|\mathcal{H}| = \sum_c \mathcal{H}(c)$. Let $color(p)$ denote the color of pixel p. The histogram for position (i, j) of I is defined as follows.

Definition 3 $\mathcal{H}_{I[i,j]}(c) = \Big|\{(i', j') \mid color(I[i + i', j + j']) = c;\ \sqrt{i'^2 + j'^2} \leq r\}\Big|$.

Note that it is easy to obtain $\mathcal{H}_{I[i,j+1]}$ from $\mathcal{H}_{I[i,j]}$ incrementally in time $\mathcal{O}(r)$. To see this, note that when the $\mathcal{H}_I$ is translated by one pixel to the right, exactly (because of the discrete nature of the digital circle) $2r + 1$ new pixels of I will come inside the circle, and exactly $2r + 1$ pixels of I will go out the circle.

We need also a histogram $\mathcal{H}_P$ for P. This is slightly more complicated to obtain than $\mathcal{H}_I$. Assume that P is at location $((u, v), \theta)$ on top of I. Consider the matching function at that location. It may happen for e.g. pixels $p = P[q, s]$ and $p' = P[q + 1, s]$ that $M(p) = M(p')$. In this case, if the colors of p and p' are different, then there cannot be an exact match at location $((u, v), \theta)$, because the color $M(p)$ cannot agree with both the colors of p and p'. This means that k must be at least one for an approximate match. On the other hand, if the colors of p, p' and $M(p)$ are the same, then a single pixel of I matches exactly with two pixels of P. This means that we may get different histograms for different angles θ. If the color of p has already been counted, then the color of p' should be ignored. Let P be at location $((u, v), \theta)$ such that $(u, v) \in I[i, j]$. Let q' and s' be such that $M(P[q, s]) = I[q', s']$. Let $g_{q',s'}$ denote the number of pixels of P of color c, that map to the pixel $I[q', s']$ when P is in location $((u, v), \theta)$. Now the

histogram of P for location $((u,v),\theta)$ is

$$\begin{aligned}\mathcal{H}_{P((u,v),\theta)}(c) &= \sum_{q,s}\{1/g_{q',s'} \mid color(P[q,s]) = c;\\ &\quad M(P[q,s]) = I[q',s'];\\ &\quad \sqrt{(i-q')^2+(j-s')^2} \le r\}.\end{aligned}$$

In some situations we may need a different kind of histogram. Let $\hat{g}$ be 1, if only one pixel p of P maps to $I[q',s']$. More precisely, if two pixels p_0 and p_1 of P map to $I[q',s']$, then let $\hat{g}_{q',s'}(p_0) = 0$ and $\hat{g}_{q',s'}(p_1) = 1$ (or vice versa). That is, we count only the color of one of the pixels of P that map to the same pixel of I. The choice is arbitrary.

$$\begin{aligned}\hat{\mathcal{H}}_{P((u,v),\theta)}(c) &= \sum_{q,s}\{\hat{g}_{q',s'}(P[q,s]) \mid\\ &\quad color(P[q,s]) = c;\\ &\quad M(P[q,s]) = I[q',s'];\\ &\quad \sqrt{(i-q')^2+(j-s')^2} \le r\}.\end{aligned}$$

As the matching function is many–to–one mapping, the number of colors in each bin of $\mathcal{H}_P$ may change as P rotates. As the correct displacement and angle of rotation is not known in the filtering phase, the exact number of colors in each bin is unknown. For some distance functions we need to use "minimum–histogram" for P:

Definition 4 $\mathcal{H}_P^{\min}(c) = \min\{\mathcal{H}_{P(\theta)}(c)\}$.

Definition 5 $\hat{\mathcal{H}}_P^{\min}(c) = \min\{\hat{\mathcal{H}}_{P(\theta)}(c)\}$.

In some other situations, a maximum histogram is needed:

Definition 6 $\mathcal{H}_P^{\max}(c) = \max\{\mathcal{H}_{P(\theta)}(c)\} = \mathcal{H}_{P(0)}(c)$.

In some cases, the mass $m(\mathcal{H})$ of the histogram $\mathcal{H}$ is useful.

Definition 7 *The mass of histogram* $\mathcal{H}$ *is* $m(\mathcal{H}) = \sum_c c\mathcal{H}(c)$.

$\mathcal{H}_P^{\max}$ can be computed in time $\mathcal{O}(|\mathcal{H}_P^{\max}|)$. $\mathcal{H}_P^{\min}$ and $\hat{\mathcal{H}}_P^{\min}$ histograms can be computed in time $\mathcal{O}(r^3)$.

4. Histogram filtering

Our filter compares $\mathcal{H}_{I[i,j]}$ and $\mathcal{H}_P$ at each position (i,j) of I. In the following sections we consider different distance (similarity) functions d, and derive efficient algorithms for them.

4.1. Hamming distance

For Hamming distance $d(p) = 0$ if $color(p) = color(M(p))$, and 1 otherwise. Now $\mathcal{H}_P^{\min}$ tells for each color the minimum number of pixels of that color that is at least required for a complete match. The difference $b(i,j)$ between $\mathcal{H}_{I[i,j]}$ and $\mathcal{H}_P^{\min}$ gives a lower bound of the number of mismatches at position (i,j) of I. This is computed as the number of colors missing from $\mathcal{H}_P^{\min}$:

$$b(i,j) = \sum_{c\in\Sigma} \max\{\ 0, \mathcal{H}_P^{\min}(c) - \mathcal{H}_{I[i,j]}(c)\ \}.$$

If $b(i,j) \le k$, then there may be an approximate occurrence of P centered at $I[i,j]$ with Hamming distance at most k. Note that b can be calculated incrementally also, in the same way as histograms $\mathcal{H}_{I[i,j]}$, in time $\mathcal{O}(r)$. That is, when the histogram is updated with new incoming (outgoing) pixel color, the sum is updated for that pixel value, in time $\mathcal{O}(1)$.

To get a good filtration capacity, the radius r must be chosen such that the expected $b(i,j)$ is greater than k. At minimum, the number of pixels inside the circle must be greater than k. This gives approximately $\pi r^2 > k \Rightarrow r > \sqrt{k/\pi}$. If $r = \mathcal{O}(k^{1/2})$, the filtering takes time $\mathcal{O}(k^{1/2}|I|)$. To get expected $\mathcal{O}(k^{1/2}|I|)$ total time, the number of verifications must be few. Note, however, that using even larger r, the filtering capability grows, so the number of verifications shrinks. Note that the filtration capacity grows as the size of the alphabet grows. To get a lower bound, assume binary alphabet. For binary alphabet, $b = |\mathcal{H}_P(0) - \mathcal{H}_I(0)|$. The expected value of b is $\frac{1}{3}\pi r^2$, where πr^2 is (approximately) the maximum value for one bin in the histograms. Our expected case analyses are based on the uniform Bernoulli model, i.e., a pixel has a given color with probability $1/\sigma$ independently of other pixels. This gives that using $r = \mathcal{O}(k^{1/2})$ should filter out most of I.

For the verification, we may use e.g. a brute force algorithm that takes time $\mathcal{O}(m^5)$, or a more sophisticated method that takes $\mathcal{O}(m^3)$ worst case time, or $\mathcal{O}(k^{3/2})$ expected time [8]. We have the following result for finding all Hamming distances $\le k$.

Theorem 1 *Assuming uniform distribution of colors, and that* $k < (\pi/4)m^2 e^{-\pi m^2/(4|\Sigma|)}$*, the k–threshold Hamming distance problem can be solved in expected time* $\mathcal{O}(|I|k^{1/2})$.

Proof. To get pessimistic bound, we simplify the situation as follows. The probability p that some pixel in $\mathcal{H}_P^{\min}$ does not match with *any* pixel in $\mathcal{H}_I$ is $(1-1/\sigma)^{|\mathcal{H}_I|}$, and $|\mathcal{H}_P^{\min}| \approx |\mathcal{H}_I| \approx \pi r^2$. Assume that each pixel in $\mathcal{H}_P^{\min}$ is independently in $\mathcal{H}_I$ with probability p, then b has binomial distribution $Bin(|\mathcal{H}_P^{\min}|, k)$ with expected value $\mu = |\mathcal{H}_P^{\min}|p \approx \pi r^2\ (1-1/\sigma)^{\pi r^2} \approx \pi r^2 e^{-\pi r^2/\sigma}$. Assume now that $k < \mu$.

Let $\mathcal{X}$ be the number of mismatching pixels in $\mathcal{H}_I$. The expected number of verifications is therefore $Pr(\mathcal{X} \le$

$k)\ |I|$. To get fast overall time, we require that

$$Pr(\mathcal{X} \leq k)\ |I|\ k^{3/2} < |I|\ k^{1/2} \Rightarrow Pr(\mathcal{X} \leq k) < 1/k.$$

The propability $Pr(\mathcal{X} \leq k) < kPr(\mathcal{X} = k)$, because we assumed that $k < \mu$, so it suffices to show that $Pr(\mathcal{X} \leq k) < 1/k^2$. Let $h = |\mathcal{H}_I|$.

$$\begin{aligned} Pr(\mathcal{X} = k) &= \binom{h}{k} p^k (1-p)^{h-k} \\ &= \frac{h^h \sqrt{h}\ p^k (1-p)^{h-k}}{k^k \sqrt{k}(h-k)^{h-k}\sqrt{2\pi(h-k)}}\ \mathcal{O}(1) \\ &= \frac{h^h\ p^k (1-p)^{h-k}}{k^k (h-k)^{h-k}}\ \mathcal{O}(1) \\ &= \left(\frac{p^{k/h}(1-p)^{1-k/h}}{(k/h)^{k/h}(1-k/h)^{1-k/h}}\right)^h \mathcal{O}(1) \\ &= a^h\ \mathcal{O}(1), \quad a < 1 \mid k < \mu \end{aligned}$$

□

If we just want to find the highest similarity peak, we set $r = m$, and run the algorithm in time $\mathcal{O}(|I|m)$.

4.2. Delta distance

In [7] *delta distance* was defined. In the delta distance, pixel p of P should match the interval of pixel values defined by the pixel $M(p)$ and its eight neighbors. The interval is spanned by the minimum and maximum pixel values in this neighborhood.

Formally, we define the interval $\mathcal{I}(i,j)$ of I at position (i,j) as follows. Let $lo(i,j) = \min\{I[i+i', j+j'] \mid i', j' \in \{-1,0,1\}\}$ and $hi(i,j) = \max\{I[i+i', j+j'] \mid i', j' \in \{-1,0,1\}\}$, and let $\delta \in \Sigma$ be a given parameter that is used to adjust the interval for tighter or looser approximation.

Definition 8 *The interval $\mathcal{I}(i,j)$ at position (i,j) of I is $\mathcal{I}(i,j) = [lo(i,j) - \delta, hi(i,j) + \delta]$.*

We may now compute the Hamming distance such that the color of pixel p of P is matched against the interval corresponding to pixel $M(p)$. That is, $d(P[i,j]) = 0$, iff $P[i,j] \in \mathcal{I}(i',j')$, and 1 otherwise, where $M(P[i,j]) = I[i',j']$. The delta distance can be thought of as another Hamming distance measure, with alphabet size σ/Δ. In the uniform Bernoulli model of probability, the average of Δ is $\overline{\Delta} = (4/5)\sigma$, which means alphabet size $\overline{\sigma} = 5/4$. In practice $\overline{\Delta}$ is much smaller, because in natural images the local variation of the colors is often small.

The delta distance can be advantageous in many applications. It tolerates small systematic distortions in the pixel values, while allowing to set the threshold number k of mismatches low, see [7].

In order to derive a histogram filter for this distance measure, we use two histograms from I, one that corresponds to values of $lo(i,j)$, and the other that corresponds to $hi(i,j)$. Let us denote those by $\mathcal{H}_I^{lo}$ and $\mathcal{H}_I^{hi}$. In order to compute b, let us define $\mathcal{H}_I'^{lo}$ as follows.

$$\begin{aligned} \mathcal{H}_I'^{lo}(0) &\leftarrow \mathcal{H}_I^{lo}(0) \\ \mathcal{H}_I'^{lo}(c+1) &\leftarrow \mathcal{H}_I^{lo}(c+1) + \max\{\mathcal{H}_I'^{lo}(c) - \hat{\mathcal{H}}_P^{\min}(c), 0\} \end{aligned}$$

$\mathcal{H}_I'^{hi}$ is defined in similar way. Now define b^{lo} as follows.

$$b^{lo}(i,j) = \sum_c \max(0, \hat{\mathcal{H}}_P^{\min}(c) - \mathcal{H}_I'^{lo}(c)).$$

Again, b^{hi} is defined in similar way. The actual distance bound is the maximum of b^{lo} and b^{hi}:

$$b(i,j) = \max(b^{lo}, b^{hi}).$$

We have the following result for finding all delta distances $\leq k$.

Theorem 2 *Assuming uniform distribution of colors, and that $k < (\pi/4)m^2 e^{-\pi m^2/(4|\Sigma|/\Delta)}$, the k–threshold delta distance problem can be solved in expected time $\mathcal{O}(|I|(\sigma + k^{1/2}))$.*

Proof. As the proof of Theorem 1. □

4.3. Sum of absolute differences

When computing $\mathcal{H}_{P((u,v),\theta)}$ in the Hamming distance case, if two pixels of P that had different color, mapped to the same pixel of I, the histogram bins of P for these colors were both incremented by one. This means that at least one of the two pixels mismatches. This pixel of P can be thought to be matched against some pixel of I, that is different from the pixel which the pixel was actually mapped to. This can be done, as the cost of the mismatch is always one, regardless of the actual color values. This is not true for other distance measures. If two pixels of P map to the same pixel of I, then only one or the other histogram bin of P should be incremented, as we do not know against what pixel value the other would be compared to. This means that we must use $\hat{\mathcal{H}}_P^{\min}$.

For general distance functions, the histogram matching can be reduced to weighted bipartite graph matching [19]. We form a complete bipartite graph $G = (V_P, V_I)$ from the histograms of P and I as follows. Create a node $v \in V_P$ for each pixel in $\hat{\mathcal{H}}_P^{\min}$, and a node $w \in V_I$ for each pixel in $\mathcal{H}_{I[i,j]}$. Note that $|V_P| \leq |V_I|$. For each node $v \in V_P$, create an edge (v,w) for each $w \in V_I$. The edge (v,w) has a weight corresponding to the distance between the colors of v and w. This can be e.g. the absolute difference of the two

colors. The lower bound distance $b(i,j)$ is now the minimum weight matching of G. Let $w(G)$ denote the weight of the minimum matching of G. Now

$$b(i,j) = w(G).$$

For general weights, there exists a $\mathcal{O}(|V|^3)$ (where $|V| = |\mathcal{H}_I| + |\hat{\mathcal{H}}_P^{\min}|$ is the number of nodes in G) time algorithm for computing $b(i,j)$. This is far too slow for our purposes. However, there is $\mathcal{O}(|V|)$ time algorithm for a special case of the problem, that assumes that the nodes are homeomorphic to either a line or a circle, and the cost function is given by the Euclidean distances along the curve [1]. In our case, this means that we can use distance $d(p) = |color(p) - color(M(p))|$. The $\mathcal{O}(|V|)$ time algorithm requires that the points (pixel colors) in the curve (line) are given in sorted order. Using the histogram, the input can be generated in time $\mathcal{O}(|\mathcal{H}_I| + \sigma)$. (For small $|\mathcal{H}_I|$, one may forget the histograms, and sort the input in time $\mathcal{O}(|\mathcal{H}_I| \log |\mathcal{H}_I|)$.)

The $\mathcal{O}(|V|)$ bound in [1] is both best and worst case, whereas for our algorithm it is the worst case. Before entering the graph matching algorithm, we modify the input histograms as follows: for each pair of bins $\hat{\mathcal{H}}_P^{\min}(c), \mathcal{H}_I(c)$, we subtract the smaller bin from the larger, and set the smaller bin to 0. This can be done, because those colors would be matched against each other anyway, with cost 0. Therefore, in many cases $|V| \ll |\mathcal{H}_I|$.

The filter runs in $\mathcal{O}(|I|(r^2 + \sigma))$ worst case time.

We have the following Conjecture for finding all locations of I, where the sum of absolute differences is $\leq k$.

Conjecture 1 *For small k the k–threshold sum of absolute differences problem can be solved in expected time $\mathcal{O}(|I|(k/\sigma + \sigma))$.* □

Our experimetal results give evidence for this Conjecture.

It is possible to make the filter faster, with some loss of sensitivity. The idea is to compute the mass of the histograms. Consider $d(p) = |color(p) - color(M(p))|$. If we require that $D \leq k$, then the following is true:

$$m(\mathcal{H}_I) - k - (|\mathcal{H}_I| - |\hat{\mathcal{H}}_P^{\min}|)\,\sigma \leq m(\hat{\mathcal{H}}_P^{\min}) \leq m(\mathcal{H}_I) + k.$$

This fact leads to a new filter that is easy to incrementalize. The filter needs only the sum of pixel colors in the histograms. Now the distance bound b is

$$\begin{aligned} b' &= \max\{0, m(\mathcal{H}_I) - m(\hat{\mathcal{H}}_P^{\min}) - (|\mathcal{H}_I| - |\hat{\mathcal{H}}_P^{\min}|)\sigma\} \\ b'' &= \max\{0, m(\hat{\mathcal{H}}_P^{\min}) - m(\mathcal{H}_I)\} \\ b &= \min\{b', b''\} \end{aligned}$$

This filter has some useful properties that are utilized in Sec. 5. This can be used for the two previous problems also, but the filtering capability would be poor in most cases. The filter runs in time $\mathcal{O}(|I|r)$.

Note that for binary alphabets ($\sigma = 2$) the Hamming distance and the sum of absolute differences are the same problems. Also, the filters for these two problems reduce to the mass–filter in the case of binary alphabets.

4.4. Sum of squared differences

In the sum of squared differences (SSD for short) the distance function is $d(p) = (color(p) - color(M(p)))^2$. The graph matching algorithms would result very poor performance when applied to SSD ($\mathcal{O}(r^6)$ time). However, we now show how to reduce the SSD minimization filtering problem to the sum of absolute differences filtering problem.

The key question is "how large $b_{sad} = \sum |color(p) - color(M(p))|$ can be, in order to get $b = b_{ssd} = \sum (color(p) - color(M(p)))^2 \leq k_{ssd} = k$"? Note that b is minimized when the differences $|color(p) - color(M(p))|$ are all of equal size. We get $b = \sum k'^2 = k'^2 |\hat{\mathcal{H}}_P^{\min}| \leq k$, where k' is the difference between color values. From this we get the sum of absolute differences problem; as there are $|\hat{\mathcal{H}}_P^{\min}|$ pixels, and the corresponding difference for each is at most k', the total maximum sum of differences is

$$k_{sad} = |\hat{\mathcal{H}}_P^{\min}| k' = |\hat{\mathcal{H}}_P^{\min}| \sqrt{k/|\hat{\mathcal{H}}_P^{\min}|} = \sqrt{k|\hat{\mathcal{H}}_P^{\min}|}.$$

Theorem 3 *The SSD filtering problem can be reduced to sum of absolute differences filtering problem, with parameter $k_{sad} = (k|\hat{\mathcal{H}}_P^{\min}|)^{1/2}$.* □

This leaves the choice of r (and hence $|\hat{\mathcal{H}}_P^{\min}|$) open.

4.5. Cross–correlation

In this section we show how to use $\mathcal{H}_P$ for filtering out locations of low correlation between P and I.

As we are now maximizing, we use $\mathcal{H}_P^{\max}$ To get an upper bound for the correlation, we maximize the product of the colors in the two histograms. To achieve this, let us define h_P as follows:

$$h_P(u) = \{\min c \mid \sum_j^c \mathcal{H}_P^{\max}(j) \geq u\}.$$

Define $h_{I[i,j]}$ for $\mathcal{H}_{I[i,j]}$ in similar way. Now the bound for the similarity can be computed as follows:

$$b(i,j) = \sum_u h_P(u)\; h_{I[i,j]}(u)$$

This is trivially computed in time $\mathcal{O}(\sigma)$. The radius r has to be set to its maximum, as we are maximizing, that is, $r = (m-1)/2$.

Conjecture 2 *For large k, the k–threshold correlation maximization problem can be solved in expected time $\mathcal{O}(|I|(\sigma + |P|^{1/2}))$.* □

Our experimetal results give evidence for this Conjecture.

In the template matching context, cross correlation is not a very useful method without normalization (contrary to the difference minimization methods).

4.6. Normalized distances

In some applications it is useful to compute normalized distances. In below, we show how to define and compute normalized versions b_n of b. For the Hamming distances the normalization does not affect the filter, as b is just scaled by a constant factor. The normalized versions of b are trivial to compute in the given time bounds in each case.

Hamming/delta distance: To get $0 \leq b_n \leq 1$ use $b_n = b/|\mathcal{H}_P^{\min}|$, or $b_n = b/|\hat{\mathcal{H}}_P^{\min}|$, respectively.

Sum of absolute differences: To get normalized distance bound, use [4]: $b_n = |b - m(\hat{\mathcal{H}}_P^{\min}) + m(\hat{\mathcal{H}}_I)|$, where $\hat{\mathcal{H}}_I$ is the histogram of the pixel colors that were matched against $\hat{\mathcal{H}}_P^{\min}$ in the graph matching algorithm.

SSD: The normalized SSD is defined as $b_n = b_{sad} \,/\, (\sum_c c^2 \hat{\mathcal{H}}_I(c))^{1/2}$.

Cross correlation: The normalized cross correlation is defined as $b_n = b \,/\, (\sum_c c^2 \mathcal{H}_I(c))^{1/2}$.

5. Speed–up by the crawling boundary

Imagine that $b(i,j) > k$, so we shift $\mathcal{H}_I$ to position $(i+1, j)$. But if $b(i,j) \gg k$, then it might be possible to use a shift that is larger than only one pixel position. That is possible because we may be able to in advance tell that $b(i+1,j) \not\leq k$, by updating only $2r+1$ pixel values. In general, if $b(i,j)$ is high enough, we deduce that $b(i+a,j) \not\leq k$, for some a. Note that this is invariant to the direction of the shift, if $b(i+a,j) \not\leq k$, then also $b(i,j+a) \not\leq k$, and $b(i+a_0, j+a_1) \not\leq k$, for some $a_0^2 + a_1^2 < a^2$. We formalize this notion, and derive fast algorithms utilizing it.

Let us assume that $s = s(b(i,j))$ gives the shift for histogram position (i,j). That is, after computing $b(i,j)$, we may shift $\mathcal{H}_I$ to any direction at most distance s. There are some subtleties in computing s, since it may actually depend on the direction of the shift, e.g. the shift along a diagonal differs somewhat from the shift along a coordinate axis. However, the variation is bounded by a multiplicative constant, and it will not affect the time bounds of our algorithms. For simplicity of presentation, and without loss of generality, we assume that s does not depend on the direction of the shift, that is, s can be tought of as a radius of a circle. The computation of s depends also on d.

As $\mathcal{H}_I$ can be shifted $\mathcal{O}(s)$ pixels to any direction, it means that by a single computation of b, it is possible to prune $\mathcal{O}(s^2)$ positions of I. We now describe the algorithm that handles the bookkeeping of the pruned area efficiently, by maintaining a boundary between the two areas. Then we describe for different choices of d, how to efficiently compute b when the shifts are longer than one pixel position.

5.1. Marking the pruned positions

The most straightforward method to prune the uninteresting positions of I, is to have a bit–matrix of the same size as the image, and simply set the bit for every position pruned. However, there are two major drawbacks. The same positions would be very likely marked many times, and secondly we cannot directly jump to the next position not yet pruned, but we must scan the matrix to find that position. This would make the algorithm slow even in the best case. Also, the method would require $\mathcal{O}(|I|)$ space.

Instead, we propose a method that keeps track of the *boundary* of the area already pruned. The boundary will divide the search space in two.

Definition 9 *Let $e = e[1..n]$ be an array of row numbers of I, such that $e[i] = j$, iff rows $j \ldots n-1$ are pruned.*

The pruning is done as follows. Assume that we are in position (i,j) in I, and that the histogram comparison tells that we may safely move $s+1$ pixels to any direction, that is, we may prune using a circle of radius s. Let $y(x) = \lfloor \sqrt{s^2 - x^2} \rfloor$ for $-s \leq x \leq s$. Now for each x set

$$e[i+x] \leftarrow j - y(x),$$

iff $e[i+x] < j + y(x)$. Otherwise, leave $e[i+x]$ untouched. In practice $y(x)$ is precomputed and stored to a table. The next position to jump can be e.g. $(i+s+1, e[i+s+1])$.

See Fig. 3 for an example. Note that the width of the boundary is the width of the image I, so the boundary is not necessarily contiguous. The advantage is that the boundary is simple to index. Pruning means now moving the boundary ("down") and the next position to be checked can be an arbitrary position in the boundary.

The disadvantage of the method is that we cannot prune all the positions if we are very deep in some local hole in the boundary (because the boundary 'function' is 'bijective'), See Fig. 4. This problem could be solved by using contiguous boundary, but it would be more time consuming to index. On average we should be able to prune approximately

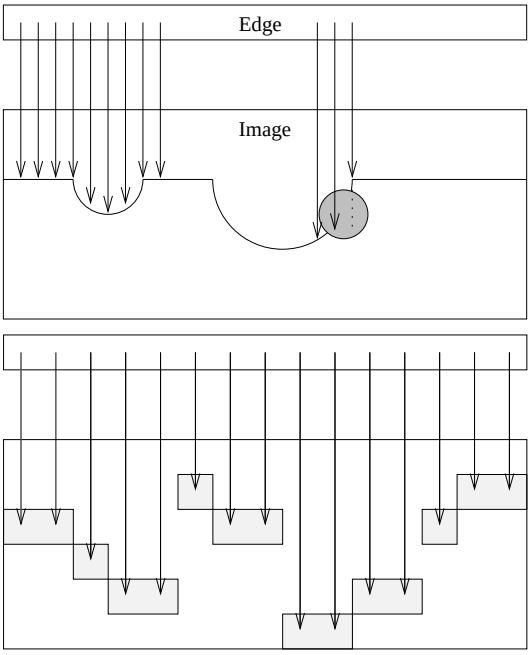

Figure 3. Pruning and boundary representation.

$\pi s^2/2$ pixels at a time at each non-matching position not yet pruned. The pruning takes time proportional to the length of the pruning boundary, which is $2s + 1$.

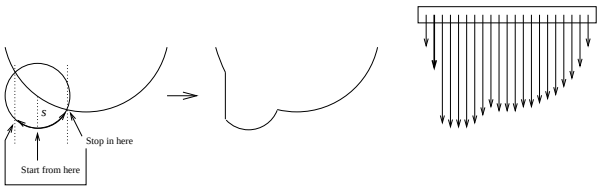

Figure 4. Pruning using circle of radius s.

5.2. Time bounds using pruning

As $\mathcal{H}_I$ is shifted $\mathcal{O}(s)$ pixels, about $\mathcal{O}(sr)$ new pixels will enter $\mathcal{H}_I$, and exactly the same number of pixels will leave $\mathcal{H}_I$. The time for updating $\mathcal{H}_I$ is therefore $\mathcal{O}(sr)$. Updating $m(\mathcal{H}_I)$ can be done in time $\mathcal{O}(r)$, regardless of s. However, this requires $\mathcal{O}(|I|)$ time and space preprocessing of the image, as follows. Compute the row–wise cumulative mass for each pixel $I[i, j]$:

$$cum(i, j) = cum(i, j - 1) + color(I[i, j]),$$

where $cum(i, 0) = 0$. Now the mass of any convex object is trivial to compute in time proportional to its height, using cum.

The time bounds given below show that the mass filter is the fastest method. However, the filtering capability is also lower. Therefore, the other method can be used as a second order filter, in the image areas where the mass filter fails. Generally, the time bounds are divided by s, or by s^2 for the mass filter, if we fix r.

Theorem 4 *By using the boundary, we achieve a speed–up factor $\mathcal{O}(1/s)$, or factor $\mathcal{O}(1/s^2)$ for the mass filter.* □

Below we give the exact time bounds for each distance/similarity measure, along with the bounds for the shifts.

In the time bounds below, it seems that minimizing r would give fast algorithms. However, this would make the expected s also small. So to maximize the filtering capability, one may just maximize r, and expect large s. Using the boundary method never makes the algorithm asymptotically slower, than without using it. So regardless of r, the shift is $s \geq 1$.

Hamming distance. As $\mathcal{H}_I$ is shifted $\mathcal{O}(s)$ pixels, about $\mathcal{O}(sr)$ new pixels will enter $\mathcal{H}_I$, and exactly the same number of pixels will leave $\mathcal{H}_I$. The total time is therefore $\mathcal{O}((|I|/s^2)(sr + s)) = \mathcal{O}(|I|r/s)$. If $b > k$, then at least $b - k$ new pixels are needed for $\mathcal{H}_I$, which gives that the shift is $s = \mathcal{O}((b - k)/r)$.

Delta distance. For the delta distance we get time $\mathcal{O}((|I|/s^2)(\sigma + sr + s)) = \mathcal{O}(|I|(\sigma/s^2 + r/s))$. As for the Hamming distance, if $b > k$, then at least $b - k$ new pixels (intervals) are needed for $\mathcal{H}_I$, which gives that the shift is $s = \mathcal{O}((b - k)/r)$.

SAD / SSD. By using the graph matching algorithm, we achieve the time bound $\mathcal{O}((|I|/s^2)(r^2 + \sigma + sr + s)) = \mathcal{O}(|I|((r^2 + \sigma)/s^2 + r/s))$. If $b > k$, then at least $(b - k)/\sigma$ new pixels are needed for $\mathcal{H}_I$, which gives that the shift is $s = \mathcal{O}((b - k)/(\sigma r))$.

With mass filter we get time $\mathcal{O}((|I|/s^2)(r + s)) = \mathcal{O}(|I|r/s^2)$. The shift in this case is $s = \mathcal{O}(\min\{s', s''\})$, where $s' = (m(\mathcal{H}_P) - m_{\min})/(\sigma r)$, and $s'' = (m_{\max} - m(\mathcal{H}_P))/(\sigma r)$, where $m_{\min}$ and $m_{\max}$ are the minimum and maximum masses allowed for $m(\mathcal{H}_P)$ by the mass–filter.

Fast thresholded circular average. In some applications it is useful to compute a circular average for each image position, that is, to compute $m(\mathcal{H}_I)/(\pi r^2)$. Assume that it is possible to set some threshold value k for the average, such that it is required that $a = m(\mathcal{H}_I)/|\mathcal{H}_I| \geq k$ (or $a \leq k$) for the actual value to be of any interest. In this case, the pruning method can be used to solve the problem in expected time $\mathcal{O}(|I|r/s^2)$.

Assuming that $a < k$, the shift is $s = \mathcal{O}((k - m(\mathcal{H}_I)/|\mathcal{H}_I|)/(r\sigma/|\mathcal{H}_I|))$.

6. Experimental results (preliminary)

We have implemented the algorithms of Sec. 4, using `C`–language, and `gcc 2.95.2` compiler, with optimization flags `-O3 -fomit-frame-pointer`. The experiments were run on 700MHz PentiumIII machine, with Linux operating system.

For the experiments we used the images shown in Fig. 5. The "correlation" images for some distance measures are shown in Fig. 6. The pixel values show how similar the corresponding image position and the template are; the brighter the color, the higher the similarity. The correlation images may be useful us such, as the highest peak seems to be in correct position, so verification may be ignored. This may not be true for all inputs.

Figures 7 and 8 show the performance of the algorithms. The filtering capability is reasonably high. Fig. 7 shows also that not only r affects the filtering capability, but that the template itself affects it; increasing r may in some cases worsen the filtering capability. The reason for this is that the histogram looses all spatial information of the pixels.

Most of the algorithms are also fast. Sum of absolute differences requires a lot of time compared to the other methods, but the timing curve looks more like linear (the best case) in r, instead of quadratic, which is the worst case behavior. The graph matching code is much more complex than the code for the other algorithms, also in terms of constant factors.

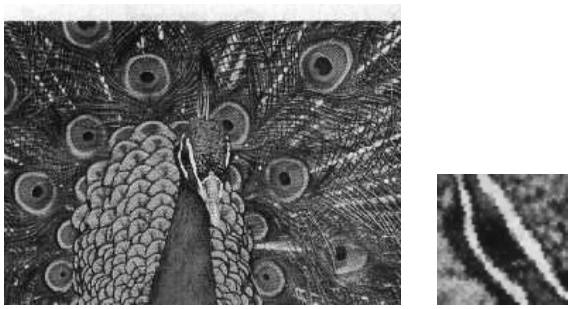

Figure 5. Peacock image (562×426) and a rotated template (49×49, not in scale). The size of the alphabet is $\sigma = 256$.

7. Conclusions and future work

We have presented fast rotation invariant filtering algorithms for locating the position of the given template pattern from large image. The algorithms work for several

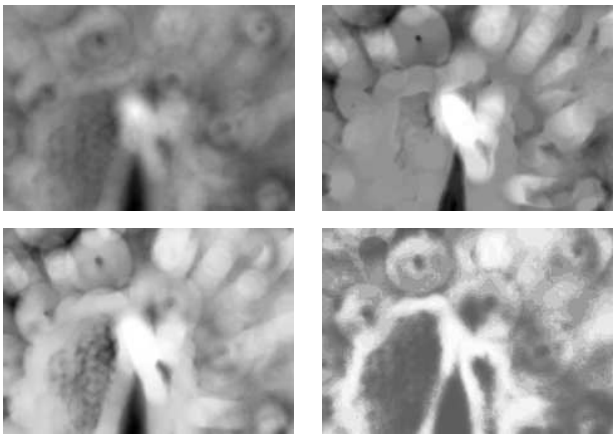

Figure 6. Peacock "correlation" images. From left to right, up to down: Hamming distance, delta distance, sum of absolute differences, and cross correlation (normalized). All images are generated using $r = (m + 1)/2$.

distance/similarity measures. The time complexity results show that difference minimization is easier than similarity maximization. The algorithms are fast and easy to implement, they give useful correlation information as such, and can be used as filtering algorithms for more sophisticated methods. Generalizations to 3D, or to other similarity measures, e.g. correlation coefficient, are straight–forward.

The filtering time for the mass–filter essentially depends on how fast the mass $m(\mathcal{H}_I)$ can be computed, which is $\mathcal{O}(r)$. However, if the circle for $\mathcal{H}_I$ is approximated using an octagon that encloses the circle, the mass–computation can be done in time $\mathcal{O}(1)$ with little loss of sensitivity, using similar methods as in [11].

Our filtering methods do not give the orientation of the template. The algorithms are, however, easy to generalize for orientation filtering also. The idea is to take two histograms from P, at distance r' from each other. If the other one is found at position (i, j) with at most distance k, then search the other one from the perimeter of a circle of radius r', centered at (i, j).

In traditional signal processing approach to rotation invariant template matching, the pixels of P are usually compared against interpolated pixel values of I. The interpolation is done because the discrete grids of pixels do not coincide perfectly. Our delta distance method can be easily generalized to handle the filtering problem for interpolated pixel values.

We believe that the sum of absolute differences (and hence also SSD) problem can be solved much faster than suggested by our experiments. The fact that the graph is formed from two histograms may be utilized to achieve

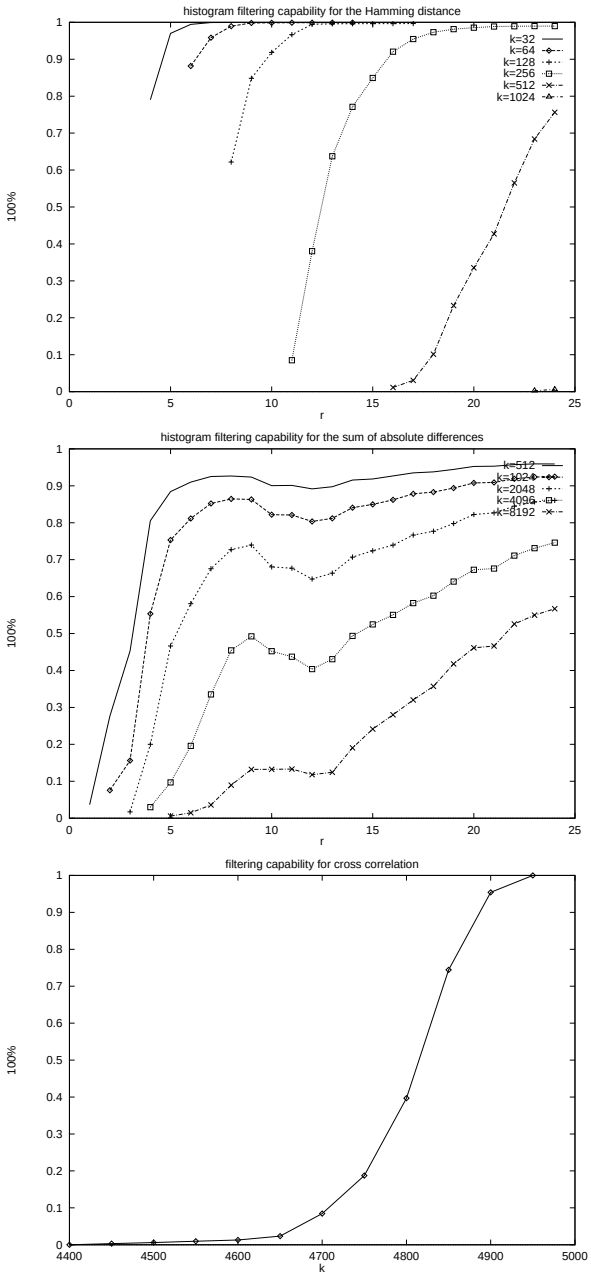

Figure 7. Filtering capability. The x–axis is r, and the y–axis is the filtering capability. Different curves are for different k. Normalized distances.

faster algorithm.

We are currently working on all these problems.

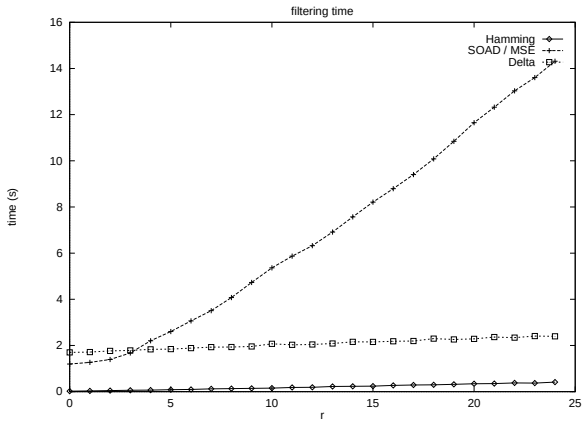

Figure 8. Running times. The x–axis is r, and the y–axis is the running time. Different curves are for different distance measures. Normalized cross–correlation runs in time $2.7s$.

References

[1] A. Aggarwal, A. Bar-Noy, S. Khuller, D. Kravets, and B. Schieber. Efficient minimum cost matching using quadrangle inequality. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 583–592, Pittsburgh, PN, Oct. 1992. IEEE Computer Society Press.

[2] A. Amir. Multidimensional pattern matching: A survey. Technical Report GIT-CC-92/29, Georgia Institute of Technology, College of Computing, 1992.

[3] A. Amir, G. Benson, and M. Farach. Alphabet independent two dimensional matching. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, pages 59–68, Victoria, B.C., Canada, May 1992. ACM Press.

[4] L. G. Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24(4):325–376, Dec. 1992.

[5] K. Fredriksson, G. Navarro, and E. Ukkonen. An index for two dimensional string matching allowing rotations. In *IFIP TCS2000*, 2000. To appear.

[6] K. Fredriksson and E. Ukkonen. A rotation invariant filter for two–dimensional string matching. In *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching*, Lecture Notes in Computer Science, pages 118–125. Springer-Verlag, Berlin, 1998.

[7] K. Fredriksson and E. Ukkonen. Combinatorial methods for approximate image matching under translations and rotations. *Pattern Recognition Letters*, 20(11–13):1249–1258, 1999.

[8] K. Fredriksson and E. Ukkonen. Algorithms for 2–D hamming distance under rotations. Manuscript, 2000.

[9] K. Fredriksson and E. Ukkonen. Combinatorial methods for approximate pattern matching under rotations and translations in 3D arrays. In *SPIRE2000*, 2000. (These proceedings).

[10] Z. Galil and K. Park. Truly alphabet-independent two-dimensional pattern matching. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages

247–257, Pittsburgh, PN, Oct. 1992. IEEE Computer Society Press.

[11] C. A. Glasbey and R. Jones. Fast computation of moving average and related filters in octagonal windows. *Pattern Recognition Letters*, 18(6):555–565, 1997.

[12] R. Grossi and F. Luccio. Simple and efficient string matching with k mismatches. *Inf. Process. Lett.*, 33(3):113–120, 1989.

[13] J. Kärkkäinen and E. Ukkonen. Two and higher dimensional pattern matching in optimal expected time. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 715–723, Arlington, VA, Jan. 1994. ACM Press.

[14] G. M. Landau and U. Vishkin. Pattern matching in a digitized image. *Algorithmica*, 12(4/5):375–408, Oct. 1994.

[15] G. Navarro and R. Baeza-Yates. Fast multi-dimensional approximate string matching. In *Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching (CPM'99)*, LNCS 1645, pages 243–257, 1999.

[16] M. Swain and D. Ballard. Color indexing. *IJCV: International Journal of Computer Vision*, 7(1):11–32, 1991.

[17] T. Takaoka. Approximate pattern matching with grey scale values. In *Proceedings of Conference on Computing: The Australian Theory Symposium*, pages 196–203, Townsville, Jan. 29–30 1996. Australian Computer Science Communications.

[18] J. Tarhio. A sublinear algorithm for two-dimensional string matching. *Pattern Recognition Letters*, 17, 1996.

[19] M. Werman, S. Peleg, and A. Rosenfeld. A distance metric for multidimensional histograms. Technical Report CAR-TR-90, University of Maryland, Center for Automation Research, 1984.

[20] J. Wood. Invariant pattern recognition: a review. *Pattern Recognition*, 29(1):1–17, 1996.