# Fast Template Matching Using Boxes Approximation and The Index Rejection Scheme

Dafna Hirschfeld

December 14, 2014

**Abstract**

This thesis presents a fast template matching algorithm. The template is represented by a linear combination of simple box functions. Fast rejection scheme is then used which bounds the correlation between the template and candidate image windows. A set of experiments shows a speed up in running time compared to common FFT-based algorithms.

## 1 Introduction

This thesis presents a fast algorithm for template matching, i.e. finding windows in an image similar to a template image. A template is a fairly small image. Given a template $T$, an image $I$ and a similarity measure or distance metric, *template matching* is finding one or more windows in the image which are most similar to the template in terms of the given metric or similarity measure. These windows are called the matches of the template. See Gonzalez and Woods [12] and Brunelli [5] for surveys of this topic.

Template matching is a basic operation used in many computer vision applications. For example, Efros and Freeman [9] and Liang et al. [16] give an algorithm for generating a large image by stitching together small patches - they use template matching in order to find regions suitable for overlap. Baker and Kanade [2] and Freeman et al. [10] construct resolution enhancement algorithms, using template matching to find suitable patches from a training dataset. Terwilliger [28] uses template matching in order to find specific locations in medical images. Tan et al. [27] uses template matching for the intra prediction method - a method for reducing the coded information of a video.

For each application an appropriate distance or similarity should be chosen. Commonly used measures are the sum of square differences (SSD), the sum of absolute differences (SAD) and normalized cross correlation (NCC). SSD and SAD are often chosen due to their simplicity. SSD is appropriate for Gaussian noise and can be computed using FFT while SAD is more robust to outliers. The drawbacks of SSD and SAD is their sensitivity to illumination change and high noise. NCC is the dot product of the template with the window after normalization: substructing the mean of the template and the window and

dividing by their norms. NCC is therefore invariant to affine transforms such as illumination change. Also, it is quite robust to Gaussian noise. Both NCC and SSD measures are computed by convolving the template with the image. The convolution gives the dot product of the template with every window in the image. If the template $T$ is of size $k \times k$ and the image $I$ is of size $n \times n$, the complexity of the direct computation of the convolution is $O(n^2k^2)$. A better way of implementing the convolution, both in theory and in practice, is to use the FFT algorithm and the convolution theorem. FFT-based convolution is of complexity $O(n^2 \log(n))$.

For certain templates the convolution can be efficiently implemented using Integral Images (Crow [6]). The integral image $II(x, y)$ of image $I(x, y)$ is:

$$II(x, y) = \sum_{i \leq x, j \leq y} I(i, j).$$

The integral image can be calculated with one scan of the image. After computing the integral image, the sum of each window takes four operations for any window size. If a window $W$ in the image $I$ has row range $r_1 \dots r_2$ and column range $c_1 \dots c_2$, then the sum over the window: $\sum_{\substack{r_1 \leq i \leq r_2 \\ c_1 \leq j \leq c_2}} I(i, j)$ can be computed using the integral image by

$$\sum_{\substack{r_1 \leq i \leq r_2 \\ , c_1 \leq j \leq c_2}} I(i, j) = II(r_2, c_2) - II(r_2, c_1 - 1) - II(r_1 - 1, c_2) + II(r_1 - 1, c_1 - 1)$$

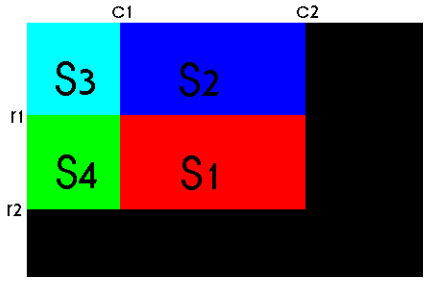



Figure 1: Computing sum over a window using Integral Image

Figure 1 gives an illustration, $II(r_2, c_2)$ is the sum over all windows: $S_1 + S_2 + S_3 + S_4$. $II(r_2, c_1 - 1)$ is the sum over the windows $S_4 + S_3$. $II(r_1 - 1, c_2)$ is the sum over the windows $S_2 + S_3$ and $II(r_1, c_1)$ is the sum over $S_3$. It is easy to verify that the above equation gives the sum over $S_1$.

This way, the convolution with a constant template can be done with 4 operations per pixel. Viola et al. [30] used Integral Image in order to extract simple features used for fast face detection. Heckbert [13] showed that convolving with polynomials of degree $d - 1$ can be done using $d$ integral images independently of the template size. Werman [32] showed an extension of this method for any

template that satisfies a linear homogeneous equation, for example, polynomials, exponentials and trigonometric sequences. Simard et al. [26] used Integral Image in order to implement a fast approximation to template matching by partitioning the template into non-overlapping blocks and approximating each block by it's mean. Briechle and Hanebeck [4] approximated the template with a linear combination of boxes functions, then the cross-correlation of the template with a window is the linear combination of the correlation of each box function with the window. They used the Integral Image in order to compute the correlation of a window with each box. In this thesis I developed a very similar algorithm that combines this method with the indexes rejection scheme described later.

For template matching, there is no need to calculate the exact dot product of the template with each of the windows in the image. This is because one is only interested in finding the windows that have the smallest distance or greatest similarity with the template. A common method to accelerate the calculations in template matching is to use the index rejection scheme. This scheme is applied for an image $I$ and a template $T$, and finds all image windows with matching score greater than a given threshold $t$. Let $S$ be the set of indexes of the candidates for the match. At the beginning all windows are candidates, so $S$ is the set of the indexes of all of the windows in $I$ of $T$'s size. At each iteration (or every few iterations) of the algorithm, an upper bound on the similarity $f$ between the template and each window whose index is in $S$ is calculated. Let $U_d(i,j)$ be the upper bound for index $(i,j)$ in iteration $d$. If $U_d(i,j) < t$ then $(i,j)$ is removed from $S$.

A General algorithm with similarity function $f$ using the indexes rejection scheme is as follow:

**Algorithm 1** Indexes-Rejection-Scheme$(I, T, t)$

Initialize $S$ to be the set of all windows $W$ in the image $I$.
$d \leftarrow 1$
**repeat**
  For each window $W$ in $S$, derive an upper bound $U_d(T,W) > f(T,W)$
  if $U_d(T,W) < t$ then $S \leftarrow S \backslash \{W\}$
  $d \leftarrow d + 1$
**until** stopping condition
For each window $W$ in $S$, compute $f(T,W)$
Output the window that gives the maximum $f$ value.

The indexed rejection scheme can also be used when $f$ is a distance function. In this case a lower bound $L_d(W,T) < f(W,T)$ is calculated on the window $W$ and the window is rejected if $L_d(W,T) > t$.

Some algorithm based on the indexes rejection scheme that use the SAD or SSD distance functions are described in [14], [3],[11] [29]. Performance evaluation of the runtime on a wide range of different inputs and theoretical complexity analysis of some of the algorithm is described in Ouyang et al. [23].

Schweitzer et al. [25] used the Cauchy-Schwarz inequality and the Walsh transform in order to derive lower and upper bounds for each window. They introduced an algorithm that compares the bounds of different windows in each iteration until the best match is found. Hel-Or and Hel-Or [14] and Ben-Artzi et al. [3] introduced an algorithm that at each iteration projects the image and the template on a basis vector in order to get tighter lower bounds on the dot products and eliminate windows. They chose basis vectors such that the successive projection and calculation of lower bounds is efficient. Mahmood and Khan [18] computed the autocorrelation of the template or the image and used the transitivity property of the convolution in order to derive the bound. Mahmood and Khan [19] manipulated the correlation equation in order to derive the upper bound. Di Stefano and Mattoccia [7], used the Cauchy-Schwartz inequality and Mattoccia et al. [21] and Di Stefano et al. [8] used Jensen inequality in order to derive bounds on the dot products. The same authors - Mattoccia et al. [21] later developed the Enhanced Bounded Correlation algorithm that uses the Cauchy Swarz inequality in order to derive a tight bound on the NCC. Their bound is derived as follow: They partitioned the template $T$ and the window $W$ into fixed $n$ non-overlaping blocks $T_1..,T_n, W_1,..,W_n$. At iteration $d$ the upper bound is:

$$UB_d(T,W) = \sum_{i=1}^{d} < T_i, W_i > + \sum_{i=d+1}^{n} \|T_i\|\|W_i\|$$

Therefore the cross-correlation is processed only on one block at each iteration. Mori and Kashino [22] and Wei and Lai [31] used a similar method. At iteration $d$ they partitioned the template $T$ and the window $W$ to $n_d$ blocks $T_1..,T_{n_d}, W_1,..,W_{n_d}$ where $n_d$ increases from iteration to iteration. The bound they used is:

$$UB_d(T,W) = \sum_{i=1}^{n_d} \|T_i\|\|W_i\|$$

As $n_d$ increases the bound gets tighter. In extreme when $n_d$ equals the number of pixels in $T$ the upper bound equals the cross correlation.

In this work I present an algorithm for the template matching problem in the following scenario. The template is given in advance and analyzed in a precomputation stage. I then use the results of the precomputation in order to run a fast template matching on a large database of images in real time. This setting is similar to learning algorithms where the training stage can take a longer time, while the classification is done in real time. This is a common situation. In many vision applications there is a need to find a certain object in a large database of images. The template is an 'interesting' patch of image - it is noiseless and it is a representative, non ambiguous part of an object - a salient object. In my work, I combine fast convolution using an integral image and an index rejection scheme using thresholds for fast template matching.

The rest of the thesis is organized as follows. In Section 2 I describe the precomputation on the template. In Section 3 I present a fast template matching algorithm, which is similar in spirit to those of Di Stefano and Mattoccia [7],

Mattoccia et al. [21] and Di Stefano et al. [8]. In Section 4 I present the output of the algorithm on several sample images. In Section 5, I present a generalization of the algorithm for images containing different scales. I conclude in Section 6 with some remarks and avenues for further research.

# 2 Template Approximation using Matching Pursuit

## 2.1 Representing a template using box functions

I represent the template as a sparse linear combination of simple templates so that the convolution with each simple template can be computed efficiently using a single Integral Image.

That is, given a template $T$ I want to find scalars $w_1, ..., w_N \in \mathbb{R}$ and templates $B_1, ..., B_N \in \mathbb{R}^{k \times k}$ so that:

1. $\sum_{i=1}^{N} w_i B_i \approx T$
2. The convolution $I * B_i$ is efficiently computable using an integral image.
3. The linear combination is sparse - $N$ is much smaller than $k^2$.

Due to the linearity of convolution:

$$I * T \approx I * \left( \sum_{i=1}^{N} w_i B_i \right) = \sum_{i=1}^{N} w_i (I * B_i). \tag{1}$$

The computation of the approximation $\{w_i, B_i \,|\, i = 1..N\}$ is a precomputation.

The algorithm I use to approximate the template is Matching Pursuit (Algorithm 2), developed by Mallat and Zhang [20]. It represents a given image as a sparse linear combination of a set of predefined templates, called atoms. The set of templates is called the dictionary. The algorithm iteratively generates a sorted list of templates and scalars from the dictionary which are a sub-optimal solution to the problem of sparse representation, see.

Matching Pursuit is a greedy algorithm. In each iteration it chooses the dictionary atom that best approximates the residual.

According to Werman [32], there are many different types of functions that can be convolved with an image using integral images. These functions include polynomials, trigonometric, exponentials, and the concatenation of such functions. I use the simplest function: a box. A 2-D box $B \in \mathbb{R}^{k \times k}$ is defined by 4 indexes $f_1, t_1, f_2, t_2$ such that

$$B(i,j) = \begin{cases} 1 & \text{if } f_1 \le i \le t_1 \text{ and } f_2 \le j \le t_2 \\ 0 & \text{otherwise} \end{cases}$$

**Algorithm 2** Matching Pursuit(T)

Let $\mathfrak{D} = \{D_1, D_2, ..., D_Q\}$ be a large dictionary of simple templates (atoms).
$i \leftarrow 1$
**repeat**
  find $D \in \mathfrak{D}$ that maximizes $|\langle D, T\rangle|$.
  $w_i \leftarrow \langle D, T\rangle$
  $B_i \leftarrow D$
  $T \leftarrow T - w_i B_i$
  $i \leftarrow i + 1$
**until** stopping condition
Output $\{w_i, B_i\}_i$

The set of indexes $\{(i, j) : B(i, j) = 1\}$ is called the support of the box. The main advantage of using the box representation is that only a single integral image is needed in order to compute the convolution of the image with any set of boxes.

Figure 2 is an Example of an image (right) and it's boxes approximation (left).

Figure 2: Example of approximating an image using a linear combination of boxes

## 2.2 Constructing a small but informative dictionary

The main challenge in the implementation of Matching Pursuit is to design a dictionary that is rich enough to ensure a sparse representation and small enough so that choosing good atoms does not take too long. I used a common solution which is restricting the size of the dictionary by using only a subset of all possible boxes (see Lin et al. [17]). The number of all possible boxes in $\mathbb{R}^{k\times k}$ is $O(k^4)$, and I used a subset of size $O(k^2)$ of all possible boxes. I chose boxes according to the length of the edges of their support. The length of the vertical edge is $t_1 - f_1 + 1$ and the length of the horizontal edge is $t_2 - f_2 + 1$.

I chose only boxes whose edges are power of 2 in length. I also restricted the boxes by their location so that for length $2^n$ only $1/2^n$ of edges of this length are chosen.

This gives a dictionary of size $O(k^2)$ (linear in $k \times k$).

For $l \leq k$, there are $k-l+1$ possible edges of length $l$, from which at most $(k-l+1)/l$ are chosen. Summing over lengths that are powers of 2 I get:

$$\sum_{n=0}^{log(k)} \frac{k-2^n+1}{2^n} = (k+1)\left(\sum_{n=0}^{log(k)} \frac{1}{2^n}\right) - (log(k)+1)$$

$$= 2(k+1)(1-(1/k)) - log(k) - 1 = O(k).$$

Thus the number one dimensional edges is $O(k)$. The dictionary is the set of boxes composed of the chosen vertical edges and the chosen horizontal edges so its size is $O(k^2)$. This dictionary gives a wide range of boxes of various sizes and locations while keeping the dictionary small.

# 3 An Efficient Template Matching Algorithm

The algorithm presented in this thesis uses the $NCC$ as a similarity measure. Given a template $T$ and a window $W$, the $NCC(T,W)$ is:

$$NCC(T,W) = \frac{\langle T-\mu_T, W-\mu_W\rangle}{\|T-\mu_T\|_2\|W-\mu_W\|_2}$$

Where $\mu_T, \mu_W$ are the averages over the template and over the window respectively. At each iteration an upper bound to $NCC(T,W)$ is derived which is faster to compute than $NCC(T,W)$ itself. The upper bound is derived using the Cauchy Schwarz inequality:

**Theorem 1 (Cauchy-Schwarz inequality)** *: Let $\alpha, \beta \in \mathbb{R}^n$ then*

$$\langle \alpha, \beta\rangle \leq \|\alpha\|\|\beta\|.$$

Using the Cauchy-Schwarz inequality one can derive the following:

**Lemma 1** *: Let $Y, P \in \mathbb{R}^{k\times k}$, let $B_1, .., B_N \in \mathbb{R}^{k\times k}$ , let $w_1, ..., w_N \in \mathbb{R}$ and let $1 \leq d \leq N$, then:*

$$\langle P, Y\rangle = \langle \sum_{i=1}^{d} w_i B_i, Y\rangle + \langle P - \sum_{i=1}^{d} w_i B_i, Y\rangle \leq \langle \sum_{i=1}^{d} w_i B_i, Y\rangle + \|P - \sum_{i=1}^{d} w_i B_i\| \cdot \|Y\|$$

Seting $Y = W - \mu_W$ and $P = T - \mu_T$ in this lemma gives upper bound $NCC(T,W) \leq U_d(T,W)$ for the $d$'th iteration:

$$NCC(T,W) = \frac{\langle T-\mu_T, W-\mu_W\rangle}{\|T-\mu_T\|\|W-\mu_W\|} \leq$$

$$\frac{\langle \sum_{i=1}^{d} w_i B_i, W-\mu_W\rangle + \|T-\mu_T - \sum_{i=1}^{d} w_i B_i\|\|W-\mu_W\|}{\|T-\mu_T\|\|W-\mu_W\|} = U_d(T,W)$$

Where $B_i \in \mathbb{R}^{k \times k}$ is the $i'th$ box template and $w_i \in \mathbb{R}$ is it's $i'th$ coefficient. The upper bound is the sum of two terms:

$$\frac{\langle \sum_{i=1}^{d} w_i B_i, W - \mu_W \rangle}{\|T - \mu_T\| \|W - \mu_W\|} \tag{2}$$

$$\frac{\|T - \mu_T - \sum_{i=1}^{d} w_i B_i\|}{\|T - \mu_T\|} \tag{3}$$

The first term can be written:

$$\frac{\langle \sum_{i=1}^{d} w_i B_i, W - \mu_W \rangle}{\|T - \mu_T\| \|W - \mu_W\|} = \frac{\langle \sum_{i=1}^{d-1} w_i B_i, W - \mu_W \rangle + \langle w_d B_d, W - \mu_W \rangle}{\|T - \mu_T\| \|W - \mu_W\|} \tag{4}$$

So the numerator of the first term can be iteratively updated to be the sum of the dot product with the next box and the previous upper-bound.

The second term is only a function of the template so it can be calculated in the pre-computation stage.

Computing the norms $\|W - \mu_W\|_2$ of all windows of size $k \times k$ of $I$ is also fast using an integral image, because $\|W - \mu_W\|_2^2 = \|W\|_2^2 - k^2 \mu_W^2 = -k^2 \mu_W^2 + \sum_{i,j} W^2(i,j)$. Computing the sum $\sum_{i,j} W^2(i,j)$ for each window is efficient using the integral image of the image of the squared values of $I$ (see Lewis [15]). Computing the average $\mu_W$ of each window is efficient using the integral image of $I$.

The algorithm's pseudo-code for image $I$, template $T$ and threshold $t$ is presented in Algorithm 3.

Complexity: Let $I$ be of size $n \times n$ and $T$ of size $k \times k$. The running time of computing the integral images and the norms $\|W - \mu_W\|$ for all of the windows is $O(n^2)$. For a single window, computing an upper bound takes $O(1)$ using the integral image. Let $\alpha_d$ be the fraction of windows left at iteration $d$. The running time of iteration $d$ is $O(\alpha_d n^2)$. Let $\beta$ be the fraction of windows left in $S$ after the last iteration. The running time of the last stage - computing the exact $NCC(T, W)$ for each window $W$ in $S$ - is $O(\beta k^2 n^2)$. Let $M$ be the number of iterations, then the total complexity is $O(n^2 + n^2 \sum_{d=1}^{M} \alpha_d + \beta n^2 k^2)$.

A drawback of index rejection scheme algorithms is that their running time depends on the content of the input and not only on its size. This is because the running time depends on the rate of window rejection, and this rate depends on the upper-bounds for each window in each iteration, and the upper-bounds depend on the content of the template and the windows (the values of the pixels). For example, if the template is smooth or the image has high noise then the rate of window rejection is slow. In the worst case, no window is rejected

**Algorithm 3** Boxes-Approximation-Rejection-Scheme($I, T, t$)

Compute the Integral Image $II_1$ of $I$ and the integral image $II_2$ of the image of squared values of $I$
Use $II_2$ and $II_1$ to efficiently compute the norm $\|W - \mu_W\|$ for each window in the image.
Initialize $S$ to be the set of all windows $W$ in the image $I$.
$d \leftarrow 1$
**repeat**
  For each window $W$ in $S$, use $II_1$ and the previously computed upper bound to compute upper bound $U_d(T, W)$
  if $U_d(T, W) < t$ then $S \leftarrow S \backslash \{W\}$
  $d \leftarrow d + 1$
**until** stopping condition
For each window $W$ in $S$, compute $NCC(T, W)$
Output the window that gives the maximum $NCC$ value.

and the complexity is $O(n^2(M + k^2))$ which is worse than direct computation of the convolution between $T$ and $I$.

The fraction of windows that are rejected at each iteration is sensitive to the threshold which needs to be adjusted to the noise level of the image. I show in the experiments that in practice the described algorithm is usually much faster than FFT based algorithms.

# 4 Experiments

I run the Matching Pursuit algorithm until 99% of the template energy is reached (sum of squares of the template's values). I limited the maximum number of boxes to 1000, since some images needed much more than 1000 boxes to reach 99% energy. The results of the experiments show that even about 100 boxes is enough.

It is possible to derive a lower bound on the size of the support of the boxes that are scanned during the Matching Pursuit. The algorithm scans the the dictionary from boxes of large support to boxes of small support until the lower bound on the size of the support is reached. This enables restricting the scanning to boxes whose support is larger then this bound. See Appendix A for the calculation of the bound.

Figure 3 shows an example of the approximation of a template of size $96 \times 96$. The right most image is of the original template and from left to right there are the images of the approximation for 15%, 10%, 5% and 1% error. It can be seen that for 15% the general structure of the images is already visible, then the next boxes refine the small details - (the texture of the fur for example). The 1% approximation is visually almost identical to the original template.

Figure 3: The original template (right) and it's approximations (left to right): for 15% error (60 atoms), 10% error (105 atoms), 5% (242 atoms), and 1% error (1000 atoms)

| template edge | 16 | 32 | 48 | 64 | 96 | 128 |
|---|---|---|---|---|---|---|
| Num. of templates | 97 | 190 | 251 | 273 | 117 | 72 |

Table 1: Number of templates chosen for each size

I collected 50 images from Google Images. 10 images of size 850x1200 and 40 images of size 425x600. The images were chosen to be of various sizes and both indoor and outdoor scenery. They were also chosen to contain many salient objects so interesting templates can be taken for the experiment. I chose the templates by hand. From each image I chose 20 templates of sizes: $16\times16, 32\times32, 48\times48, 64\times64, 96\times96, 128\times128$. Figure 4 shows two examples of images and some templates taken from them. Table 1 shows the total number of templates chosen for each size.

I ran the Matching Pursuit algorithm on each template. The graphs in figures 5 and 6 show the result statistics of the Matching Pursuit algorithm. The graph in figure 5 shows the average number of boxes needed as a function of the error percentage for each template size. The graph in figure 6 shows the average running time to calculate the approximation as a function of the error percentage for each template size. The graph in Figure 5 shows that on average it is enough to have 300 boxes in order to reach 5% error approximation for templates of size 128x128. This is because most natural images have most of their energy on a low frequency. For example, for 128x128 templates, the average number of boxes needed in order to get 1% error is about 900 while the number of boxes needed for 5% error is about 300, therefore 600 boxes are needed to cover a small percentage of the template energy.

I tested the template matching algorithm using the collection of 50 images and the templates extracted from them. Altogether there are 1000 templates. For each template I tested the algorithm with 3 levels of Gaussian noise with standard deviations $\sigma = 0.05, 0.1, 0.2$, (2.5, 25.5 and 50 gray levels respectively). Noise was added to the images using MATLAB's imnoise operation.

Calibrating the parameters: The algorithm's performance is sensitive to the threshold. The threshold should be set to be the maximal value that will not

Figure 4: Example of images in the experiments and templates taken from them

cause the correct match to be rejected. I wanted the model to be as simple as possible so at first I set the threshold only according to the input size and noise level. I sampled 200 templates (from images that are not one of the 50 images of

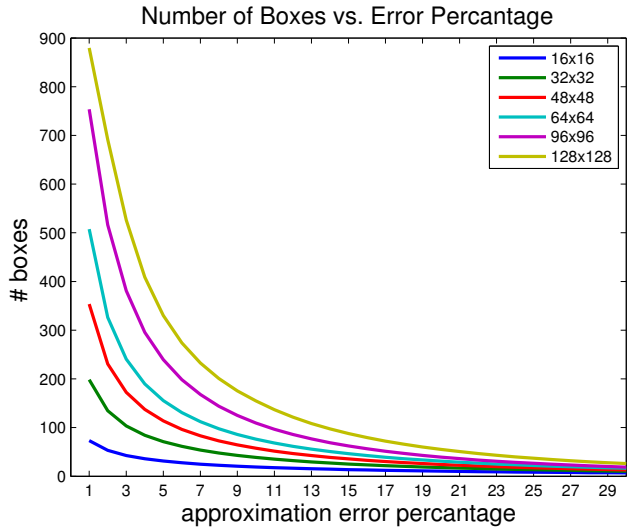



Figure 5: The number of boxes vs error percentage

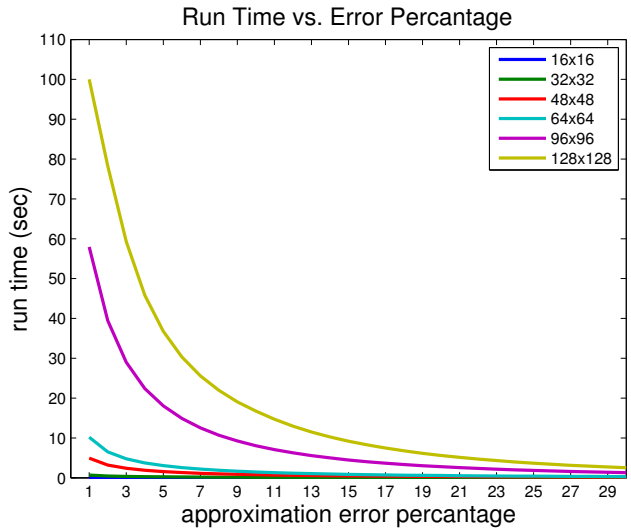



Figure 6: The run time to compute the approximation vs error percentage

the experiments). For each template and each noise level I computed the $NCC$ value of the template with the noisy version of itself, - This gives an estimation for the thresholds, and then by taking the sample mean and standard deviation of the results I tried to find a good set of thresholds that will fit the input only according to the template size and image noise level.

Unfortunately this attempt did not give good results, The problem is that I want the algorithm to be as correct as the FFT based template matching. Whenever FFT finds the right match so do I. The problem with setting a global threshold that is used for all input of the same template size and image noise is that there are templates whose $NCC$ score with their right match is very low and still FFT finds the right match for them. Therefore the global threshold must be low in order to not reject the right match of any template and thus the performance of the algorithm for most templates is bad.

Instead of a global threshold for all possible templates I set a series of thresholds for each template separately. A threshold for each template and each iteration. In the experiments the templates are noiseless, and each template is searched in a noisy version of the image it was extracted from. The noise is additive Gaussian. The upper bound of the right match is treated as a random variable. This random variable is a function of the noise on each pixel in the match. In this model, since the template and the noise distribution are known before running the matching algorithm, one can calculate the distribution of the upper-bound on the right match that results from each box. This calculation can be done in the pre-computation stage.

The threshold can be set to be small enough so that the upper-bound will most likely be larger than the threshold. Formally, Let $T$ be a template, $I$ a noisy image with Gaussian noise with standard deviation $\sigma$ and let $W$ be the window in $I$ which is the match to $T$. In this model, $W$ is the result of adding Gaussian noise to $T$. Let $U_d(T, W)$ be the upper bound for iteration $d$. Then $U_d(T, W)$ is a random variable that depends on the noise. I can choose small $\epsilon > 0$ and find $t_d$ such that $P(U_d(T, W) > t_d) > 1 - \epsilon$. That is, with probability close to 1, the upper-bound of the right match will be larger than the threshold $t_d$. If $\epsilon$ is very small, I can be almost sure that the right match won't be rejected. Using this method I have a table of thresholds for each iteration and noise level. Appendix C gives the calculation of the distribution of the upper-bound on the right match.

I implemented the the algorithm in C and compiled it with gcc 4.9.1 without optimizations. I compared the results to the matchTemplate function of the OpenCV library [1]. The stopping criteria is the number of iterations. The experiments show that very few iterations are needed in order to get optimal results. Figures 7,8,9,10 show the speedup of my algorithm with up to 1000, 50, 10 and 5 iterations respectively (These numbers of iterations are only an upper bound because the number of iterations is also bounded by the number of boxes). The speedup is the average over the ratios of the running time of OpenCV's algorithm with running time of my algorithm for each input. The standard deviation for each template size and noise is also shown in the figures. These figures show that 50 iterations are enough for optimal results for low and moderate noise. Running up to 1000 iterations didn't improve the results for input of images with low and moderate noise and only slightly improved the speedup over input of images with high noise and template size 128x128. Reducing the number of iterations from 50 to 10 or 5 reduces the performance of the algorithm. Even for 5 iterations my algorithm outperforms OpenCV for

all template sizes and image noise levels.

Running 1000 iterations neither improved nor harmed the algorithm's performance, because the majority of the calculations in the algorithm are concentrated mostly in the start and end of the algorithm. In the start: calculating the integral image and running the first few iterations where many windows are not yet rejected. In the end calculating the NCC separately on each window that was not rejected. 50 iterations gave optimal results.

Figure 11 is the graph of the average and standard deviation of the run time of my algorithm for small and large images with Gaussian noise of 25.5 pixels. The graph shows that the run time of the algorithm is almost unaffected by the template size.

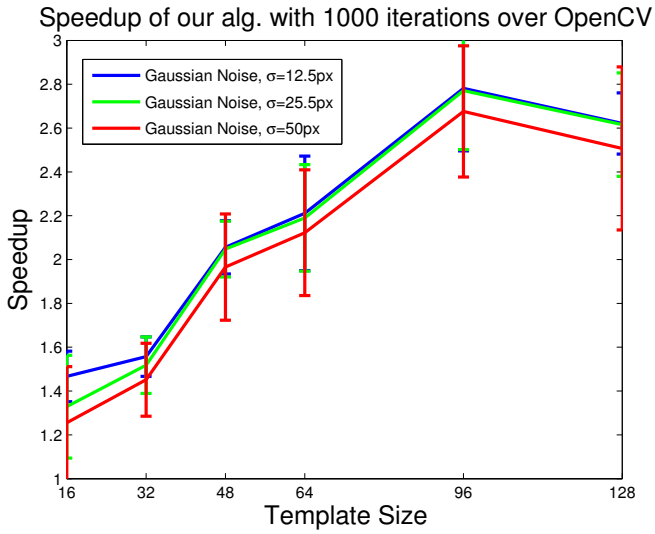



Figure 7: Speedup of the suggested algorithm with up to 1000 iterations compared to OpenCV

In section 3, I analyzed the theoretical complexity of the algorithm: $O(n^2 + n^2 \sum_{d=1}^{M} \alpha_d + \beta n^2 k^2)$. There are two things that might harm the performance of the algorithm:

1) The number of rejected windows is too small, and thus causing heavy calculations in the end of the algorithm - computing the exact NCC for each left window. This case is described by the term $\beta n^2 k^2$.

2) The algorithm might perform bad on an input if the rejection rate is low, causing heavy calculations over the iterations. The term $n^2 \sum_{d=1}^{M} \alpha_d$, is the total number of windows scanned over the iterations. High value of this term indicates that many windows were rejected toward the end of the running of the loop or not at all.

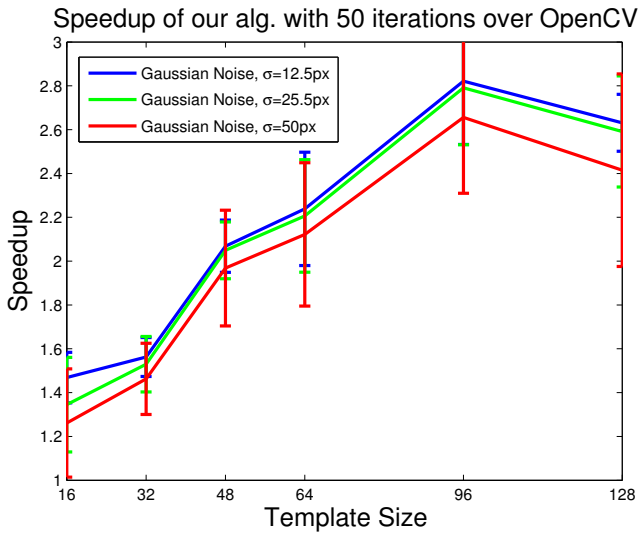



Figure 8: Speedup of the suggested algorithm with up to 50 iterations compared to OpenCV

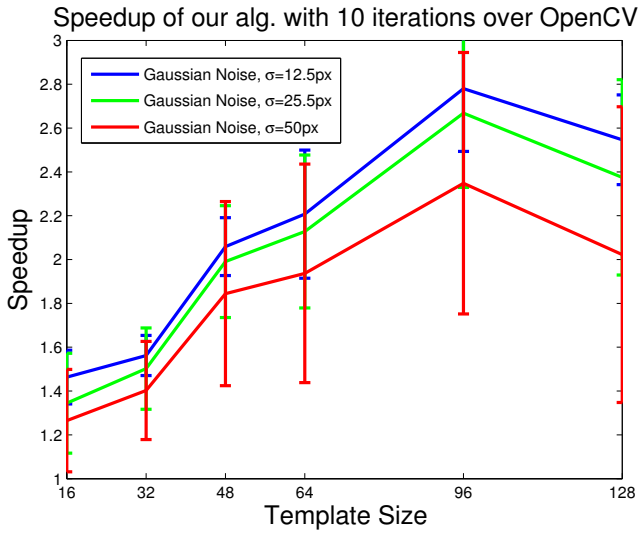



Figure 9: Speedup of the suggested algorithm with up to 10 iterations compared to OpenCV

In order to better understand the behavior of the algorithm I examined some of the inputs where the algorithm performed worse than FFT. I also examined two additional types of input: templates for which the number of rejected images

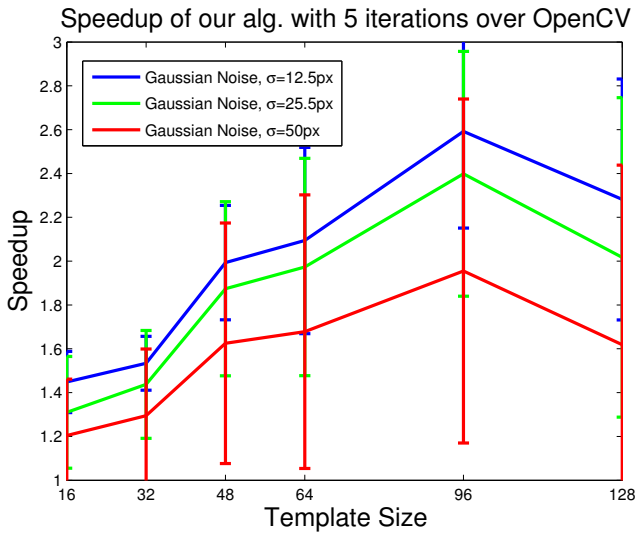



Figure 10: Speedup of the suggested algorithm with up to 5 iterations compared to OpenCV

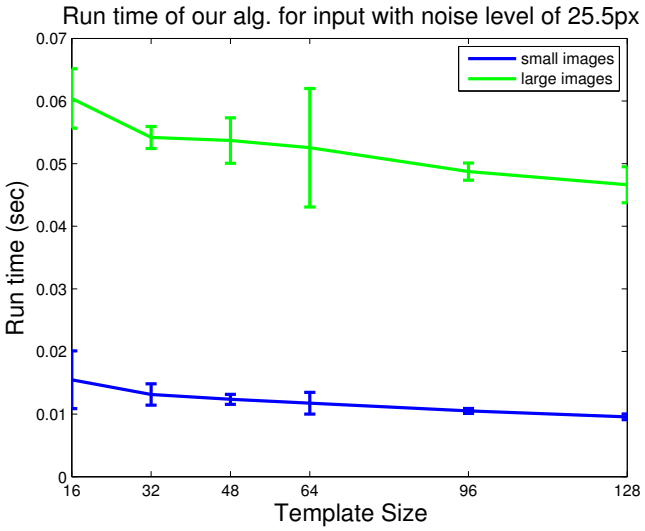
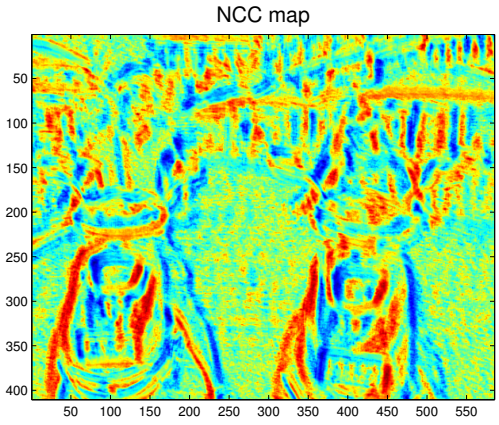


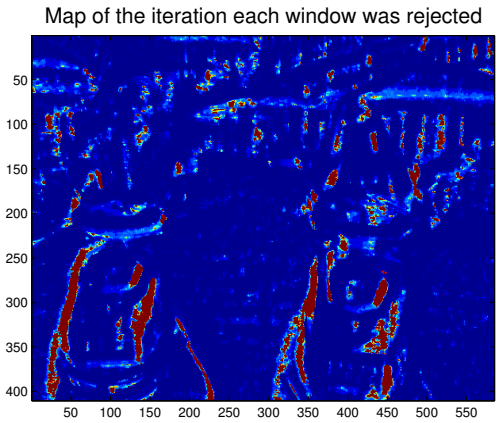

Figure 11: Average and standard deviation of the run time of the algorithm on input with noise of 25.5 pixels

was small and templates for which the total amount of windows scanned was large.


(a) The image and template (marked in red)




(b)

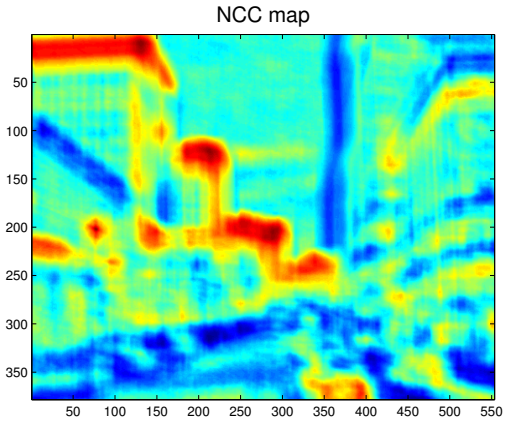




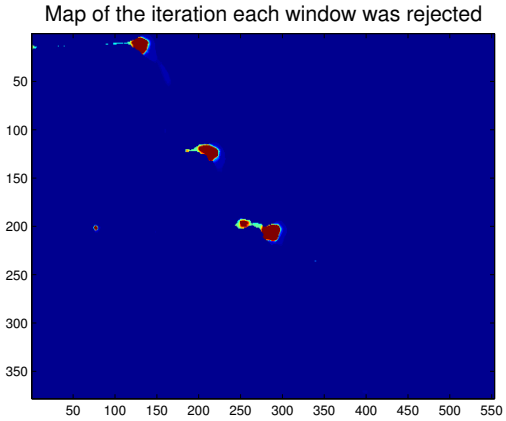

(c)

Figure 12: An input where the suggested alg. is slower than OpenCv's


(a) The image and template (marked in red)



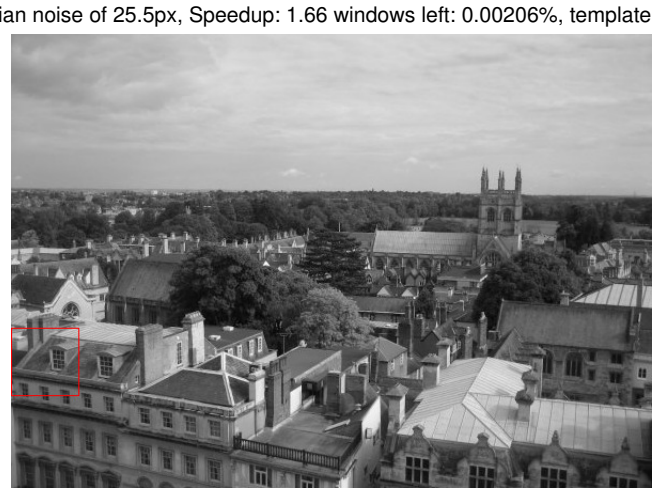

(b)



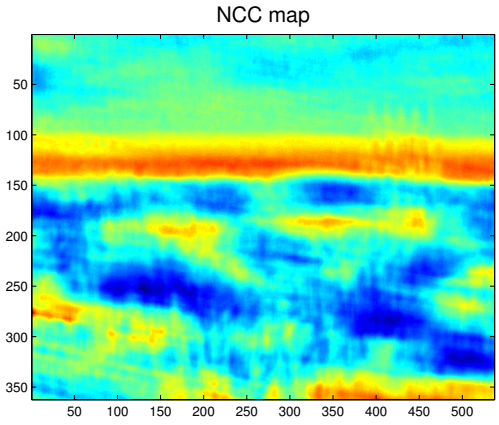



(c)

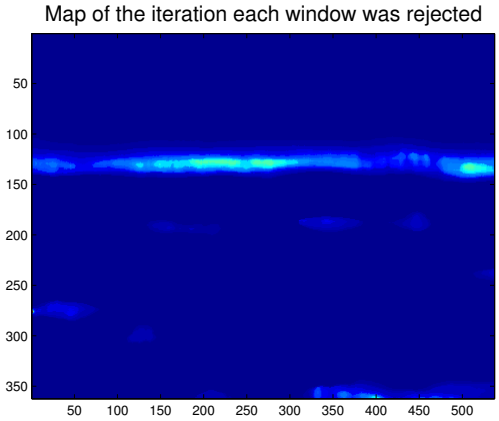
Figure 13: An input with high fraction of non-rejected windows




(a) The image and template (marked in red)




(b)






(c)

Figure 14: An input with low rejection rate

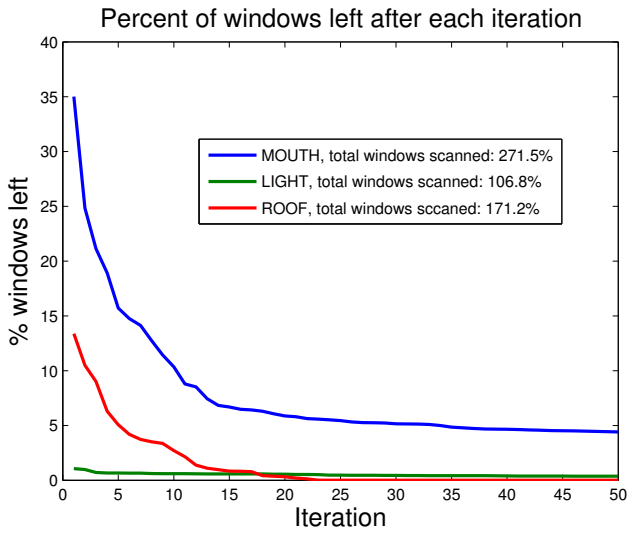



Figure 15

Figures 12, 13, 14 are 3 examples of input that I examined. In each case the noise level was 25.5 pixels and the number of iterations was 50. Figure 12 is an example of an input where the performance worse than OpenCV. The speedup on this input is 0.38 meaning OpenCV was $\frac{1}{0.38} = 2.6$ times faster than ours. Figure 13 is an example of input with a high percentage of non-rejected windows. Figure 14 is an example of input with a large amount of windows scanned. For each of the three examples I show the input itself - the image and the template in red square and two color-maps: The NCC map of the template with the image, and a rejection map - each index in this map indicates the iteration in which it was rejected. Non rejected indexes have the highest value (total number of iterations +1). Figure 15 shows a graph of the rejection rate of the 3 examples (labeled by order of appearance: MOUTH, LIGHT, ROOF). It also has on the legend the total percent of windows scanned over all iterations for each example. (This value is always larger than 100 because in the first iteration all windows are scanned).

The graph in figure 15 show that the first example suffers both from a high fraction of non-rejected windows and a low rejection rate. The second example suffers from a high fraction of windows left - 0.37%. In its rejection map 3 large areas of non rejected windows are clearly seen. In example 3 the rejection rate is slow - in the rejection map there is a line corresponding to the horizon. This line indicates a large portion of windows that were not immediately rejected. For all three examples the speedup was lower than the average speedup for their size.

Many times it is possible to predict if the performance on a certain template will be bad. It is often the case that there is not enough information in the template, so it is similar to many windows. This is the case in the examples in

figures 12 and 13. I also noticed that most inputs with bad performance are of templates size 16x16, which are small and thus have little information. In other cases the performance is bad because of the image content. For example in the case of the input example in figure 14 since it has a clear horizon line, templates that are brighter in the upper part have strong correspondence with the horizon line, and so the windows on the horizon line will be rejected only later in the rejection process. The cases where the bad performance due to image content are harder to avoid since the images are online input to the algorithm while the templates are given in advance.

Figures 16 and 17 show the histograms of the run time of the algorithm on large and small images respectively both with Gaussian noise level of 25.5 pixels. These histograms show that on most inputs the run time is in a certain range, and there are a few inputs for which the algorithm is slower.

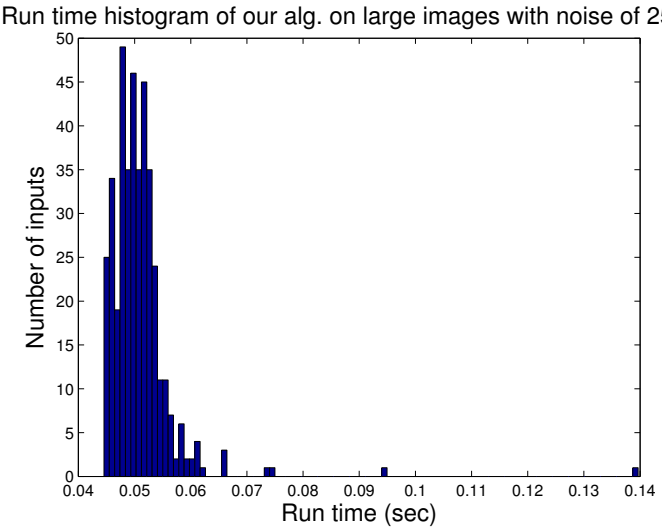



Figure 16: Histogram of the run time of the algorithm on large images with Gaussian noise level of 25.5 pixels.

In addition to the main experiment I described above, I tested the algorithm on other types of data sets. The algorithm is based on the assumption that the templates are "interesting" and noiseless - representing objects or parts of objects. I was interested to see how the algorithm behave on patches of textures, like grass or treetops. I was also interested to see how it behave on "bad" templates - templates for which the NCC does not give the right match to them.

I chose 8 images out of the 50 images from the main experiment and chose 10 patches of texture of size 64x64 from each image. Figure 18 is an example of an image with the texture patches taken from it.

For texture patches that are not smooth, the NCC map usually has high

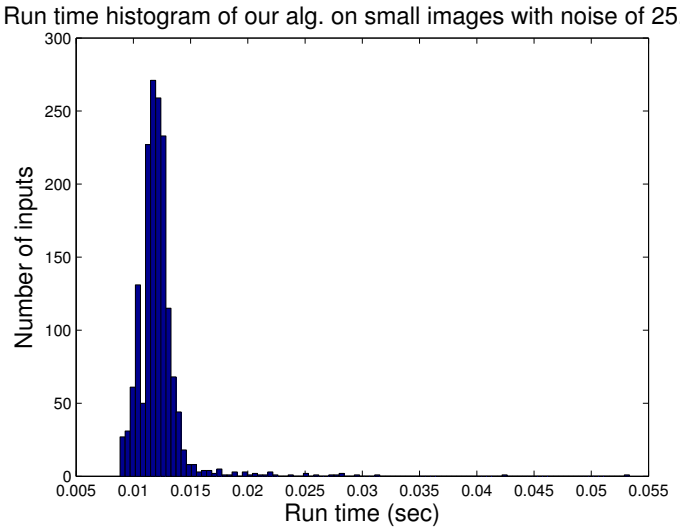



Figure 17: Histogram of the run time of the algorithm on small images with Gaussian noise level of 25.5 pixels.

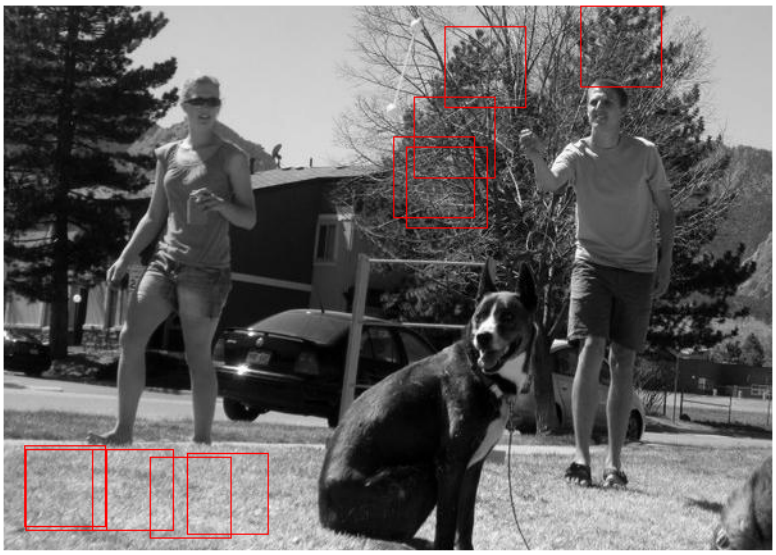


Figure 18: texture patches example

value in the location of the right match and in a narrow set of neighbors of the right match and has low value on other indexes. The speedup results of the run time of my algorithm with 50 iterations over OpevCV are summarized in table 2. The speedup for texture patches is smaller than for templates from the main experiment. Still, the suggested algorithm performed better than OpenCV.

| noise level | speedup | standard deviation |
|---|---|---|
| 12.5 px | 2.05 | 0.2 |
| 25.5 px | 1.85 | 0.56 |
| 50 px | 1.45 | 0.75 |

Table 2: Speedup results for texture patches

Another set of templates that I examined were templates where the $NCC$ measure did not give the right match. I chose templates such that the NCC measure gives the right match when the noise level is 25.5 pixels but not when the noise level is 50 pixels. This means that the templates did have information that could be distinguished with moderate noise but not enough to be distinguished with high noise. Figure 19 is an example of such a template. I measured the run time speedup over OpenCV for those templates for images with high noise. The Graph in figure 20 shows the results for data set of 55 templates of size 16x16, 41 templates of size 32x32 29 templates of size 48x48 and 12 templates of size 64x64 for small images. The graph shows that the algorithm performs worse than OpenCV for all template sizes. This indicates that the algorithm is good only for "interesting" templates.

Figure 19: Example of template for which the NCC measure doesn't give the right match for it on image of noise level of 50 pixels

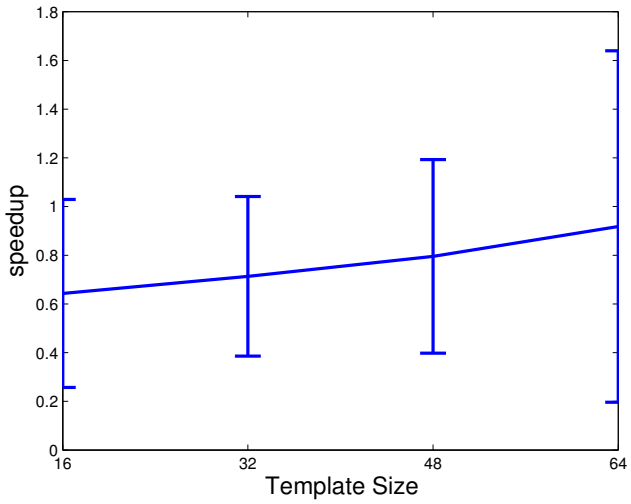



Figure 20: Speedup of the suggested algorithm over openCV for inputs for which the NCC measure doesn't give the right match for them on images of noise level of 50 pixels

In the last experiment I ran the algorithm on a small specific set of inputs. I chose 4 pairs of images of sizes 1000x700, 900x600, 800x640, 640x480 where each pair are images of the same scene with a small difference in focus / position / light / quality. From each pair I extracted 5 templates of size 96x96 from one of the images and extracted the approximated location of the corresponding window in the other image by hand. The pairs of images and templates are shown in figure 21.

I ran the algorithm with thresholds calculated the same way as in the main experiment assuming Gaussian noise of $\sigma = 0.5$. I compared the running time with openCV. For each of the algorithms the template taken from one image was searched in the other image. I decided that a correct match has been achieved if the window returned by the algorithm is in the cirular range of radius 15 from the location that I chose by hand.

I ended up with good results. There was one template for which both the algorithms missed the right match. For all of the inputs the run time was better than openCV's run time. The average run time for the templates in each image are summarized in table 3

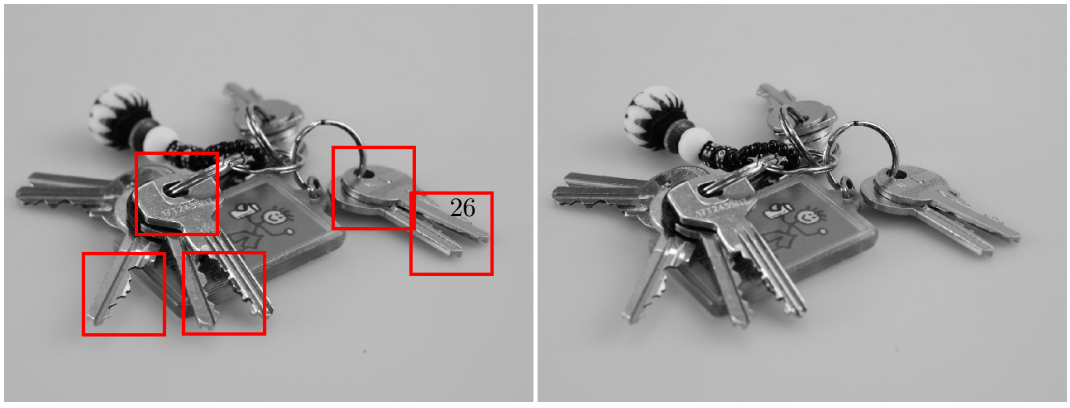
| - | Suggested alg. run time | openCV run time | Speedup |
|---|---|---|---|
| pair 1 | 0.039 | 0.102 | 2.61 |
| pair 2 | 0.027 | 0.069 | 2.55 |
| pair 3 | 0.022 | 0.069 | 3.31 |
| pair 4 | 0.018 | 0.033 | 1.83 |

Table 3: Run time results (sec) for natural data experiment

(a) pair 1

(b) pair 2

(c) pair 3



(d) pair 4

# 5 Application - Template Matching with Different Scales

In real applications of template matching it is almost always the case that the right scale is unknown and there is need for template matching with different scales. This is when the object is known, but the size of the object in the image is unknown. For example, as an object gets farther away from the camera it gets smaller in the image, so the template of the object is in a certain scale but had to be searched different scales. This is common in the case of tracking a moving object in a video. Richardson et al. [24] recently showed that for a scene in which there is a planar surface and the camera's X axis is parallel to that surface the scale of objects moving on that surface is linear in the y coordinate in the scene. Such conditions usually hold in videos recorded by surveillance cameras. In those videos the scene usually consists of people or cars moving on the flat ground, and the X axis of the camera is parallel to the ground.

Given the image $I = I(r, c)$ A scale map $SM = SM(r, c)$ can be computed such that $SM(r, c)$ is the scale assumed for the template in the $(r, c)$ location.

In case of videos recorded by surveillance cameras as mentioned, $SM(r, c)$ is a linear function of $r$ - the index of the row. A direct implementation to the scaled template matching will be: For each row, resize the template according to the scale of this row in the scale map and compute the $NCC$ value of the resized template with each window in this row. The template matching for a given row can be computed by either FFT method or direct convolution.

The representation of the template as a linear combination of boxes enables a faster implementation. Using image rejection scheme as before and scaling each box according to the scale map. Let be the approximation:

$$T(r, c) \approx \sum_{i=1}^{N} w_l B_l(r, c)$$

and let scale factor be $s \in \mathbb{R}$, then

$$T(s(r, c)) \approx \sum_{i=1}^{N} w_i B_i(s(r, c))$$

This is because scaling a linear combination of boxes is equivalent to scaling each box separately. In case $s$ is not an integer the scaling will be rounded to the nearest integer. This scaling is good for the purpose of template matching since it is very fast and avoids filtering and interpolation that are usually required in image resize.

## 5.1 Examples for Scaled Template Matching

I collected two videos of scenes of moving cars. I chose two cars from each video. For each car I ran 8 scaled template matchings : 4 template matchings where

the template is smaller than it's match in the image (the scale is larger than 1) and 4 template matchings where the template is larger than it's match in the image (the scale is smaller than 1). I implemented both algorithms (direct and ours) in MATLAB.

Figures 22 shows an example from the first video where the template is larger than the match. Figures 22a and 22b are the image and the template of the scaled template matching. Figure 22c is the match that was found by the scaled template matching by both the direct algorithm and the boxes algorithm. Figure 22d is the synthetic image results by scaling each box such that the resulted image is of the size of the found match. Figures 23 shows an example from the second video where the template is smaller than the match.


(a) The image of TM of match smaller than template

(b) The template for the TM of match smaller than template

(c) The match found

(d) The result of the box representation of the template by scaling each box

Figure 22: scaled TM example for match smaller than template

(a) The image of TM of match larger than template

(b) The template for the TM of match smaller than template

(c) The match found

(d) The result of the box representation of the template by scaling each box



Figure 23: scaled TM example for match smaller than template

Figure 24 shows the NCC map (letf) and upper-bounds map (right) for the template matching of the image and template in figures 23a and 23b, you can see the red spot corresponding to the right match in both maps.

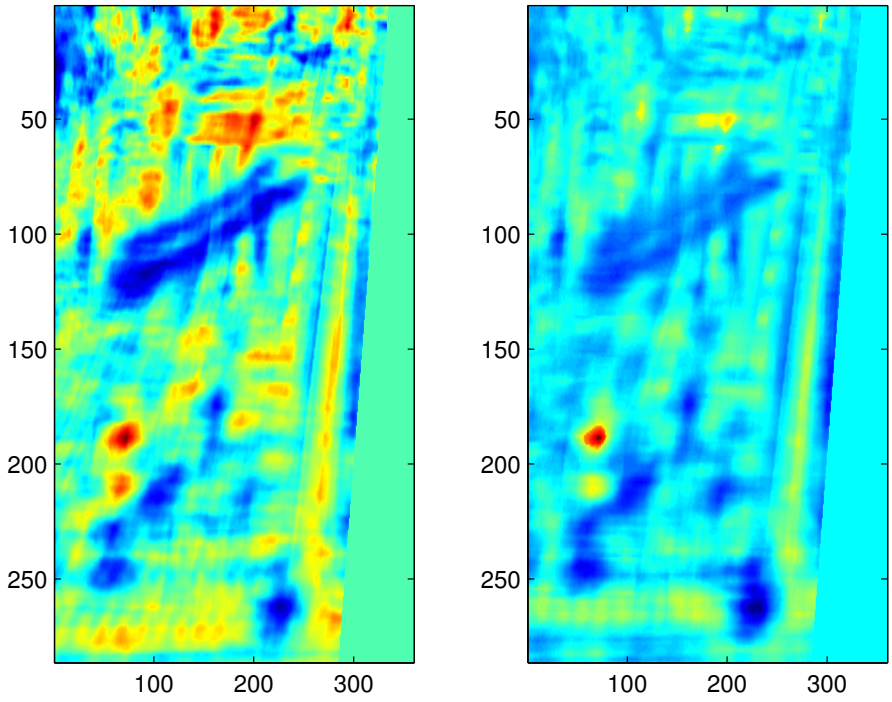



Figure 24: left: results of TM in template smaller than match. Left: NCC map. Right: upper-bounds map of the suggested algorithm

The performance of the algorithm in all examples was better than the direct implementation. In all examples I use one threshold for algorithm set to 0.9, and the number of iterations was set to 100.

| runtime(sec) | Direct alg. runtime | Suggested alg. runtime | Speedup | scale |
|---|---|---|---|---|
| video 1 ex1. | 1.82 | 0.74 | 2.46 | 0.35 |
| video 1 ex2. | 2.19 | 0.55 | 4 | 0.33 |
| video 2 ex1. | 0.72 | 0.19 | 3.6 | 0.43 |
| video 2 ex2. | 0.58 | 0.29 | 2.02 | 0.67 |

Table 4: Scaled template matching algorithms results for scale $< 1$

| runtime(sec) | Direct alg. | Suggested alg. | Speedup | scale |
|---|---|---|---|---|
| video 1 ex1. | 1.51 | 0.61 | 2.45 | 1.41 |
| video 1 ex2. | 2.19 | 0.34 | 6.35 | 2.42 |
| video 2 ex1. | 0.67 | 0.17 | 3.9 | 1.78 |
| video 2 ex2. | 0.56 | 0.21 | 2.69 | 1.28 |

Table 5: Scaled template matching algorithms results for scale $> 1$

## 6 Conclusions and Future Work

In this thesis I developed an algorithm for fast template matching using Integral Image and the indexes rejection scheme. The experiments confirm that the algorithm has good performance for meaningfull templates. The algorithm does not fit textured or smooth templates. The algorithm can be naturally extended to template matching with different scales without decrease in running time. In this work I used box functions in order to approximate the template. As a future work a more sophisticated approximation can be exemined like a combination of polynomials of different degree and cosines.

# Appendices

## A Dictionary Scanning

For a box $B$ with support $R$, the best approximation of the template $T$ with that box is to set the coefficient of the box to be the avarge value of $T$ over the range $R$, call it $\mu_{T,R}$:

$$\mu_{T,R} = \frac{1}{|R|} \sum_{(i,j)\in R} T(i,j) \tag{5}$$

The Mathcing Pursuit search the box that minimize $\|T - \mu_{T,R}B\|^2$:

$$\begin{aligned} &\arg\min_B \|T - \mu_{T,R}B\|^2 = \arg\min_B \sum_{(i,j)\in R} (T(i,j) - \mu_{T,R})^2 + \sum_{(i,j)\notin R} T(i,j)^2 = \\ &\arg\min_B \sum_{(i,j)} T(i,j)^2 + |R|\mu_{T,R}^2 - 2\mu_{T,R} \sum_{(i,j)\in R} T(i,j) = \arg\min_B |R|\mu_{T,R}^2 - 2\mu_{T,R}^2|R| = \\ &\arg\min_B(-|R|\mu_{T,R}^2) = \arg\max_B |R|\mu_{T,R}^2 \end{aligned} \tag{6}$$

Where the third equation is because the first term is independent on B and can be removed and because $\sum_{(i,j)\in R} T(i,j) = |R|\mu_{T,R}$ . The range of $T(i,j)$ is

[-1,1], therefore $\mu_{T,R}^2 < 1$, so $|R|\mu_{T,R}^2 \leq |R|$ . If the i'th box in the dictionary has range $R_i$ and $M_i = |R_i|\mu_{T,R_i}^2$, then for $j > i : |R_j| < M_i \Longrightarrow M_j < |R_j| < M_i$. So the Matching Pursuit can scan the boxes from boxes with large support to boxes with small support until a box for which the upper bound holds.

# B Setting the Thresholds

Let $T$ be the $k \times k$ template and let $W$ be the right match to the template with Gaussian noise of variance $\sigma^2$. Let $n(j) \sim \mathcal{N}(0, \sigma^2)$ be the noise of pixel $j$. That is, $W = T + n$. The upper bound $U_d(T, W)$ is a random variable. Let $0 < \epsilon$, I want to find $t_d \in \mathbb{R}$ such that $P(U_d(T, W) < t_d) < \epsilon$, so if $\epsilon$ is small enough then $t_d$ will be a good threshold. The upper bound is the sum of terms (2) and (3). Term (3) does not depend on the noise so it is a constant. Term (2) is a ratio of two random variables. I can find two different bounds on the numerator and denominator separately as follow:

Let $V$ be term (2), $P(V < t_d) < \epsilon \Leftrightarrow P(V \geq t_d) > 1 - \epsilon$. Denote $V = \frac{u}{v}$ and let $t_1 > 0, t_2 > 0$ be two constants and $E_1, E_2$ be the events $E_1 = \{u \geq t_1\}, E_2 = \{v \leq t_2\}$, If both events hold then $\frac{u}{v} \geq \frac{t_1}{t_2}$. If there are $t_1, t_2$ such that $P(\overline{E_1}) < \epsilon, P(\overline{E_2}) < \epsilon$ then using the union bound:

$$P(V \geq \frac{t1}{t2}) > P(E_1 \cap E_2) = 1 - P(\overline{(E_1 \cap E_2)}) = 1 - P(\overline{E_1} \cup \overline{E_2}) > 1 - P(\overline{E_1}) - P(\overline{E_2}) > 1 - 2\epsilon$$

So it is enough to find such $t_1, t_2$ and set $t = \frac{t_1}{t_2}$.

## B.1 Evaluating the numerator:

$$\langle \sum_{i=1}^{d} w_i B_i, W - \mu_W \rangle \tag{7}$$

The numerator is a linear combination of independent normal variables. The following lemma state that the linear combination of normal variables is also a normal variable:

**Lemma 2** *Let $X \sim \mathcal{N}(\mu_1, \sigma_1^2), Y \sim \mathcal{N}(\mu_2, \sigma_2^2)$ independent and let $a, b, c \in \mathbb{R}$ then*

$$aX + bY + c \sim \mathcal{N}(a\mu_1 + b\mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2) \tag{8}$$

$$aX - bY + c \sim \mathcal{N}(a\mu_1 - b\mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2) \tag{9}$$

Since $W = T + n$ then

$$\mu_W = \frac{1}{k} \sum_{k=j}^{k} (T(j) + n(j)) = \mu_T + \frac{1}{k} \sum_{j=1}^{k} n(j) \tag{10}$$

So the numerator is:

$$\langle\sum_{i=1}^{d} w_i B_i, W-\mu_W\rangle = \langle\sum_{i=1}^{d} w_i B_i, T+n-\mu_T-\frac{1}{k}\sum_{j=1}^{k} n(j)\rangle = \langle\sum_{i=1}^{d} w_i B_i, T-\mu_T\rangle+\langle\sum_{i=1}^{d} w_i B_i, n-\frac{1}{k}\sum_{j=1}^{k} n(j)\rangle \tag{11}$$

The first term in the sum is constant. The second term can be represented by a sum of independent normal variables. This is because the noises $n(j_1), n(j_2)$ are independent for every $j_1 \neq j_2$. I need to find the coefficient of $n(r)$ for the $r'th$ pixel. For a certain box $w_i B_i$ with support $R_i = \{m : B_i(m) = 1\}$, if $r$ is in the support of $w_i B_i$ then the coefficient of $n(r)$ in the dot product

$$\langle w_i B_i, n - \frac{1}{k}\sum_{j=1}^{k} n(j)\rangle \tag{12}$$

is $w_i(1 - \frac{|R_i|}{k})$. If $r$ is not in the support of $w_i B_i$ then the coefficient of $n(r)$ in the dot product of equation (12) is $-\frac{w_i|R_i|}{k}$. Therefore:

$$\langle\sum_{i=1}^{d} w_i B_i, n-\frac{1}{k}\sum_{j=1}^{k} n(j)\rangle = \sum_{i=1}^{d}\langle w_i B_i, n-\frac{1}{k}\sum_{j=1}^{k} n(j)\rangle = \sum_{j=1}^{k}(\sum_{i:j\in R_i} w_i(1-\frac{|R_i|}{k})-\sum_{i:j\notin R_i}\frac{w_i|R_i|}{k})n(j) \tag{13}$$

The last term in (13) is the representation of the second term in (11) as a sum of independent normal variables and Lemma 2 can be used in order to calculate it's distribution.

## B.2 Evaluating the denominator:

$$\|T-\mu_T\|\|W-\mu_W\| \tag{14}$$

$$\|W-\mu_W\| = \langle W-\mu_W, W-\mu_W\rangle = \sum_{j=1}^{k}((W-\mu_W)(j))^2 \tag{15}$$

This is a sum of squares of not independent normal variables $(W-\mu_W)(j)$, $j = 1..k$. For simplicity they will be treated as independent. This will give an approximation to the distribution which is good enough. For a single noise $n(r)$:

$$(W-\mu_W)(r) = T(r)+n(r)-\mu_t-\frac{1}{k}\sum_{j=1}^{k} n(j) = T(r)-\mu_t+n(r)(1-\frac{1}{k})-\frac{1}{k}\sum_{j\neq r}^{k} n(j) \sim \mathcal{N}(T(j)-\mu_t, \sigma^2\frac{k-1}{k}) \tag{16}$$

Therefore

$$\langle W-\mu_W, W-\mu_W\rangle = \sum_{j=1}^{k}((W-\mu_W)(j))^2 = \sigma^2(\frac{k-1}{k})\sum_{j=1}^{k}\frac{((W-\mu_W)(j))^2}{\sigma^2(\frac{k-1}{k})} \tag{17}$$

Since the variables are treated as if they are independent, the sum

$$\sum_{j=1}^{k} \frac{((W-\mu_W)(j))^2}{\sigma^2(\frac{k-1}{k})} \tag{18}$$

is a sum of squares of independent normal variables divided by their standard deviation therefore the term in (18) is distributed Noncentral-Chi-square: $\chi^2(k,\lambda)$ Where $k$ - the number of pixels is the number of degrees of freedom and $\lambda = \sum_{j=1}^{k}(\frac{T(j)-\mu_T}{\sigma\sqrt{\frac{k-1}{k}}})^2$ the noncentrality parameter. Now, $\langle W-\mu_W, W-\mu_W\rangle$ is of distribution Noncentral-Chi-Square scaled by $\sigma^2(\frac{k-1}{k})$ and $\|W-\mu_W\| = \sqrt{\langle W-\mu_W, W-\mu_W\rangle}$ is the root of scaled Noncentral-Chi-Square distribution. By manipulation on the CDF (Cumulative Distribution Function) of the Noncentral-Chi-Square distribution I can get the CDF of of the denominator. In addition I can get the inverse of the CDF in order to find a threshold $t_d$ that fits the given $\epsilon$.

# References


[1] http://opencv.org.

[2] Simon Baker and Takeo Kanade. Hallucinating faces. In *Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on*, pages 83–88. IEEE, 2000.

[3] Gil Ben-Artzi, Hagit Hel-Or, and Yacov Hel-Or. The gray-code filter kernels. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(3):382–393, 2007.

[4] Kai Briechle and Uwe D Hanebeck. Template matching using fast normalized cross correlation. In *Aerospace/Defense Sensing, Simulation, and Controls*, pages 95–102. International Society for Optics and Photonics, 2001.

[5] R Brunelli. Template matching techniques in computer vision: Theory and practice, 2009.

[6] Franklin C Crow. Summed-area tables for texture mapping. *ACM SIGGRAPH computer graphics*, 18(3):207–212, 1984.

[7] Luigi Di Stefano and Stefano Mattoccia. A sufficient condition based on the cauchy-schwarz inequality for efficient template matching. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, volume 1, pages I–269. IEEE, 2003.

[8] Luigi Di Stefano, Stefano Mattoccia, and Federico Tombari. Zncc-based template matching using bounded partial correlation. *Pattern recognition letters*, 26(14):2129–2134, 2005.

[9] Alexei A Efros and William T Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346. ACM, 2001.

[10] William T Freeman, Thouis R Jones, and Egon C Pasztor. Example-based super-resolution. *Computer Graphics and Applications, IEEE*, 22(2):56–65, 2002.

[11] Mohammad Gharavi-Alkhansari. A fast globally optimal algorithm for template matching using low-resolution pruning. *Image Processing, IEEE Transactions on*, 10(4):526–533, 2001.

[12] Rafael C Gonzalez and Richard E Woods. Digital image processing, 2002.

[13] Paul S Heckbert. Filtering by repeated integration. In *ACM SIGGRAPH Computer Graphics*, volume 20, pages 315–321. ACM, 1986.

[14] Yacov Hel-Or and Hagit Hel-Or. Real-time pattern matching using projection kernels. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(9):1430–1445, 2005.

[15] JP Lewis. Fast normalized cross-correlation. In *Vision interface*, volume 10, pages 120–123, 1995.

[16] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics (ToG)*, 20(3):127–150, 2001.

[17] Jian-Liang Lin, Wen-Liang Hwang, and Soo-Chang Pei. Fast matching pursuit video coding by combining dictionary approximation and atom extraction. *Circuits and Systems for Video Technology, IEEE Transactions on*, 17(12):1679–1689, 2007.

[18] Arif Mahmood and Sohaib Khan. Exploiting transitivity of correlation for fast template matching. *Image Processing, IEEE Transactions on*, 19(8): 2190–2200, 2010.

[19] Arif Mahmood and Sohaib Khan. Correlation-coefficient-based fast template matching through partial elimination. *Image Processing, IEEE Transactions on*, 21(4):2099–2108, 2012.

[20] Stéphane G Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *Signal Processing, IEEE Transactions on*, 41(12): 3397–3415, 1993.

[21] Stefano Mattoccia, Federico Tombari, and Luigi Di Stefano. Fast full-search equivalent template matching by enhanced bounded correlation. *Image Processing, IEEE Transactions on*, 17(4):528–538, 2008.

[22] Minoru Mori and Kunio Kashino. Fast template matching based on normalized cross correlation using adaptive block partitioning and initial threshold estimation. In *Multimedia (ISM), 2010 IEEE International Symposium on*, pages 196–203. IEEE, 2010.

[23] Wanli Ouyang, Federico Tombari, Stefano Mattoccia, Luigi Di Stefano, and Wai-Kuen Cham. Performance evaluation of full search equivalent pattern matching algorithms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(1):127–143, 2012.

[24] Eitan Richardson, Shmuel Peleg, and Michael Werman. Scene geometry from moving objects. In *Advanced Video and Signal Based Surveillance (AVSS), 2014 11th IEEE International Conference on*, pages 13–18. IEEE, 2014.

[25] Haim Schweitzer, Robert Finis Anderson, and Rui Deng. A near optimal acceptance-rejection algorithm for exact cross-correlation search. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1089–1094. IEEE, 2009.

[26] Patrice Y Simard, Léon Bottou, Patrick Haffner, and Yann LeCun. Boxlets: a fast convolution algorithm for signal processing and neural networks. *Advances in Neural Information Processing Systems*, pages 571–577, 1999.

[27] Thiow Keng Tan, Choong Seng Boon, and Yoshinori Suzuki. Intra prediction by template matching. In *Image Processing, 2006 IEEE International Conference on*, pages 1693–1696. IEEE, 2006.

[28] Thomas C Terwilliger. Automated main-chain model building by template matching and iterative fragment extension. *Acta Crystallographica Section D: Biological Crystallography*, 59(1):38–44, 2002.

[29] Federico Tombari, Stefano Mattoccia, and Luigi Di Stefano. Full-search-equivalent pattern matching with incremental dissimilarity approximations. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(1): 129–141, 2009.

[30] Paul Viola, Michael J Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 734–741. IEEE, 2003.

[31] Shou-Der Wei and Shang-Hong Lai. Fast template matching based on normalized cross correlation with adaptive multilevel winner update. *Image Processing, IEEE Transactions on*, 17(11):2227–2235, 2008.

[32] Michael Werman. Fast convolution. *Journal of WSCG*, 11(1), 2003.