# Fast pattern matching using orthogonal Haar transform

Wanli Ouyang, Renqi Zhang, Wai-Kuen Cham
The Chinese University of Hong Kong
The Chinese University of Hong Kong, Shatin N.T., HongKong
wlouyang@ee.cuhk.edu.hk

## Abstract

*Pattern matching is a widely used procedure in signal processing, computer vision, image and video processing. Recently, methods using Walsh Hadamard Transform (WHT) and Gray-Code kernels (GCK) are successfully applied for fast transform domain pattern matching. This paper introduces strip sum on the image. The sum of pixels in a rectangle can be computed by one addition using the strip sum. Then we propose to use the orthogonal Haar transform (OHT) for pattern matching. Applied for pattern matching, the algorithm using strip sum requires $O(\log u)$ additions per pixel to project input data of size $N \times N$ onto $u$ 2-D OHT basis while existing fast algorithms require $O(u)$ additions per pixel to project the same data onto $u$ 2-D WHT or GCK basis. Experimental results show the efficiency of pattern matching using OHT.*

## 1. Introduction

Pattern matching, also named as template matching, is the procedure of seeking a given pattern in given signal data. For pattern matching in images, the pattern is usually a 2D image fragment which is much smaller than the image. This procedure is shown in Fig. 1. Pattern matching is widely used in signal processing, computer vision, image and video processing. It has found applications in image based rendering [16], image compression [24], object detection [14], super resolution [17], texture synthesis [15], block matching in motion estimation [25], image denoising [11, 13], road/path tracking [6], mouth tracking [31], image matching [39] and action recognition [40]. To relieve the burden of high computational time required by pattern matching, a lot of fast algorithms have been proposed [5, 19, 8, 10, 26, 29, 33, 32].

This paper proposes a fast algorithm that computes the sum of pixels in a rectangle by one addition using the strip sum. Strip sum is used for computing OHT. When computed on sliding windows, OHT requires $O(\log u)$ additions per pixel to project input data onto $u$ basis vectors using strip sum. OHT is then used for pattern matching. Experimental results show that pattern matching using OHT is faster than existing fast *full search* (FS) equivalent pattern matching algorithms.

The paper is organized as follows. Section 2 introduces pattern matching, integral image and rectangle sum. Section 3 presents the strip sum and analyzes the computation and memory required by strip sum for computing rectangle sum. The OHT is then introduced. Section 4 gives experimental results in using OHT for pattern matching. Finally, Section 5 presents conclusions.

## 2. Related work

### 2.1. Pattern matching using transformation

Suppose an $N_1 \times N_2$ pattern is to be sought in a given image as shown in Fig. 1. The pattern will be compared with candidate windows of similar size in the image. We represent the pattern as vector $\vec{X}_t$ having length $N$ and represent the candidate windows as $\vec{X}_w^{(j)}$ having length $N$, where subscripts $\cdot_t$ and $\cdot_w$ denote template and window respectively, $j = 0, 1, \ldots, W-1$ and $N = N_1 N_2$. For example, if a $16 \times 16$ pattern is searched in a $256 \times 256$ image, we have $N = 256$ and $W = (256 - 16 + 1)^2 = 58081$. The distance between $\vec{X}_t$ and $\vec{X}_w^{(j)}$, which is denoted as $d(\vec{X}_t, \vec{X}_w^{(j)})$, is used for measuring the dissimilarity between $\vec{X}_t$ and $\vec{X}_w^{(j)}$. The smaller is $d(\vec{X}_t, \vec{X}_w^{(j)})$, the more similar are $\vec{X}_t$ and $\vec{X}_w^{(j)}$. If $T$ is the threshold that discriminates between matched and mismatched candidates, then $\vec{X}_w^{(j)}$ is considered to match $\vec{X}_t$ when $d(\vec{X}_t, \vec{X}_w^{(j)}) < T$.

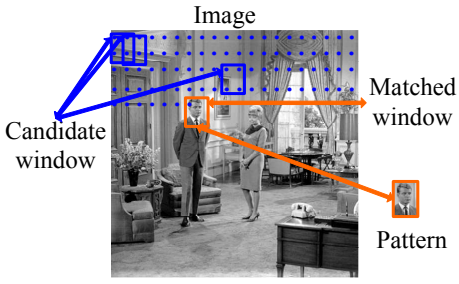


Figure 1. Pattern matching in image 'couple'.

The distance between $\vec{X}_t$ and $\vec{X}_w^{(j)}$ can be measured by $||\vec{X}_t - \vec{X}_w^{(j)}||^2$ which is also named as the sum of squared differences (SSD) between $\vec{X}_t$ and $\vec{X}_w^{(j)}$. Most transform domain pattern matching algorithms use SSD as distance measure [8, 19, 29]. As pointed out in [19], although there are arguments against the SSD as a dissimilarity measure for images, it is still commonly used due to its simplicity. Further discussion on this can be found in [18].

The transformation that projects a vector $\vec{X} \in \mathbb{R}^N$ onto a linear subspace spanned by $U$ basis vectors $\vec{V}^{(N,0)}, \ldots \vec{V}^{(N,U-1)}$ can be represented as follows:

$$\vec{Y}^{(U)} = V^{(U\times N)}\vec{X} = [\vec{V}^{(N,0)} \ \ldots \ \vec{V}^{(N,U-1)}]^T\vec{X}, \quad (1)$$

where $\cdot^T$ is matrix transposition, vector $\vec{X}$ of length $N$ is called input window, vector $\vec{Y}^{(U)}$ of length $U$ is called projection value vector, the $U$ elements in vector $\vec{Y}^{(U)}$ are called projection values and $V^{(U\times N)}$ is a $U \times N$ matrix that contains $U$ orthogonal basis vectors $\vec{V}^{(N,i)}$ of length $N$ for $i = 0, \ldots, U-1$. The transformation is called projection in [10, 19]. Basis vector is also called projection kernel in [19] and called filter kernel in [8].

As proved in [19], the following inequality holds when the $u$ basis vectors in $V^{(u\times N)}$ are orthonormal:

$$||\vec{X}_t - \vec{X}_w^{(j)}||^2 \geq ||V^{(u\times N)}\vec{X}_t - V^{(u\times N)}\vec{X}_w^{(j)}||^2. \quad (2)$$

Denote the set of candidates as $set_{can}$, which initially contains all candidates. According to (2), if $||V^{(u\times N)}\vec{X}_t - V^{(u\times N)}\vec{X}_w^{(j)}||^2 > T$, then $||\vec{X}_t - \vec{X}_w^{(j)}||^2 > T$, and so we can safely prune candidate window $\vec{X}_w^{(j)}$ from $set_{can}$. In this way, $||V^{(u\times N)}\vec{X}_t - V^{(u\times N)}\vec{X}_w^{(j)}||^2 > T$ is a sufficient condition for rejecting mismatched window. For each iteration of $u$, where $u$ increases from 1, $\vec{X}_t$ and $\vec{X}_w^{(j)}$ are projected onto transform domain using $V^{(u\times N)}$ and this sufficient condition is checked for the remaining candidates in $set_{can}$. Finally, when $u$ reaches a sufficient number, which is denoted as $N_{Maxu}$, the iteration terminates and then the remaining candidate windows in $set_{can}$ undergo FS for finding out the matched windows. The procedure of transform domain pattern matching is summarized in Table 1. Such pattern matching approach is FS equivalent. The $u$ basis vectors $\vec{V}^{(N,0)}, \ldots, \vec{V}^{(N,u-1)}$ in $V^{(u\times N)}$ are selected from the $U$ orthonormal vectors $\vec{V}^{(N,0)} \ \ldots \ \vec{V}^{(N,U-1)}$ in $V^{(U\times N)}$. For example, the strategy in [28] selects the $u$ Walsh Hadamard Transform (WHT) basis vectors having lowest frequency components.

The main advantage of using transformation is that it is more efficient computing $||V^{(u\times N)}\vec{X}_t - V^{(u\times N)}\vec{X}_w^{(j)}||^2$ than computing $||\vec{X}_t - \vec{X}_w^{(j)}||^2$ and a small number of projection values can eliminate a large number of mismatched windows. According to [19], pattern matching using WHT as the transformation is almost two orders of magnitude faster than FS and can help deal with illumination effect and multi-scale pattern matching. Sometimes, the pattern to be matched may not be rectangular, the method proposed by Ben-Yehuda et al. in [10] helps GCK and WHT to deal with this problem by segmenting the pattern into dyadic components. Because of these advantages, transform domain pattern matching has been used for block matching in motion estimation for video coding [25, 28], road/path tracking [6], mouth tracking [31], wide baseline image matching [39], texture synthesis [20] and augmented reality [38]. However, the transformation takes the main computational time in pattern matching. It is desirable to have new fast algorithms to perform the transformation.

Table 1. Transform domain pattern matching

| |
|---|
| *Overall procedure*:<br>Initially, $set_{can}$ contains all candidate windows $\vec{X}_w^{(j)}$;<br>For $u = 1$ to $N_{Maxu}$:<br>{ For $\vec{X}_w^{(j)}$ in $set_{can}$: {*Step a.1*; *Step a.2*;} }<br>The remaining candidate windows undergo FS. |
| **Step a.1**: $\vec{X}_t$ and $\vec{X}_w^{(j)}$ are projected to obtain $V^{(u\times N)}\vec{X}_t$ and $V^{(u\times N)}\vec{X}_w^{(j)}$ respectively.<br>**Step a.2**: If $||V^{(u\times N)}\vec{X}_t - V^{(u\times N)}\vec{X}_w^{(j)}||^2 > T$, then $\vec{X}_w^{(j)}$ is removed from $set_{can}$. |

A pattern matching algorithm using nonorthogonal Haar-like features is proposed in [32]. This algorithm is not FS-equivalent. Compared with the nonorthogonal Haar-like features [36, 23], it is pointed out in [8] that the Walsh Hadamard transform (WHT) is orthogonal and can extract more energy from images using the same number of operations, which makes WHT better than nonorthogonal Haar-like features in pattern matching. However, the orthogonal Haar-like features have not been studied. In this paper, we will use the OHT for FS-equivalent pattern matching.

## 2.2. Rectangle sum and integral image

Calculating the sum of pixels within a rectangle, which is named as rectangle sum in this paper, is a very common operation on images. Rectangle sums and Haar-like features have been widely used for a lot of applications such as object detection [35, 37, 34, 41], object classification [21], pattern matching [26, 33, 30], feature point based image matching [7] and texture mapping [12]. Since the work in [12] from 1984, it has been considered that at least 3 additions are required to compute the rectangle sum using the summed area table in [12, 22] or the integral image method in [35]. In this paper, subtraction is equivalent to addition regarding computational complexity.

This section illustrates the integral image following [35, 30, 22]. For $J_1 \times J_2$ image $X(j_1, j_2)$, where $0 \leq j_1 \leq J_1 - 1$ and $0 \leq j_2 \leq J_2 - 1$, the integral image $I(j_1, j_2)$ is defined as:

$$I(j_1, j_2) = \sum_{u=0}^{j_1-1}\sum_{v=0}^{j_2-1} X(u, v). \quad (3)$$

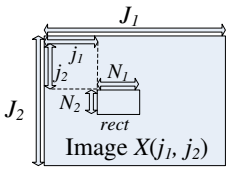


Figure 2. Rectangle $rect$ in image $X(j_1, j_2)$. $j_1$ is the horizontal position and $j_2$ is the vertical position of the upper left corner of the $rect$; $J_1$ is the width and $J_2$ is the height of the image.

It is reported by Viola and Jones that the integral image in (3) can be constructed using two additions per pixel [35]. A rectangle in the image $X(j_1, j_2)$ is specified by:

$$rect = (j_1, j_2, N_1, N_2), \tag{4}$$

where $0 \le j_1, j_1 + N_1 \le J_1 - 1; 0 \le j_2, j_2 + N_2 \le J_2 - 1$; $N_1, N_2 > 0$; $j_1$ and $j_2$ denote the upper left position of the rectangle; $N_1$ and $N_2$ denote the size of the rectangle. An example of upright rectangles are shown in Fig. 2. As pointed out in [35], the sum of pixels within the rectangle, i.e. $rect = (j_1, j_2, N_1, N_2)$ in (4), can be computed from the integral image as follows using three additions:

$$\begin{aligned} RS(rect) &= \sum_{u=j_1}^{j_1+N_1-1} \sum_{v=j_2}^{j_2+N_2-1} X(u,v) \\ &= I(j_1+N_1, j_2+N_2) + I(j_1, j_2) \\ &\quad - I(j_1, j_2+N_2) - I(j_1+N_1, j_2). \end{aligned} \tag{5}$$

Using similar method as [35], Lienhart and Maydt [22] compute the sum of pixels within the $45^\circ$ rotated rectangle by 3 additions. The summed area table in [12] is very similar to the integral image.

## 3. The fast algorithm for computing rectangle sum and orthogonal Haar transform

In this section, the fast algorithm for computing rectangle sum is introduced and analyzed. OHT and its computation are then presented.

### 3.1. Computation of rectangle sum by strip sum

Define the horizontal strip sum $HSS(j_1, j_2, N_2)$ as the sum of pixels $X(u, v)$ for $0 \le u \le j_1 - 1, j_2 \le v \le j_2 + N_2 - 1$. $HSS(j_1, j_2, N_2)$ can be constructed by one addition per pixel as follows using the integral image defined in (3):

$$\begin{aligned} HSS(j_1, j_2, N_2) &= \sum_{u=0}^{j_1-1} \sum_{v=j_2}^{j_2+N_2-1} X(u,v) \\ &= \sum_{u=0}^{j_1-1} \sum_{v=0}^{j_2+N_2-1} X(u,v) - \sum_{u=0}^{j_1-1} \sum_{v=0}^{j_2-1} X(u,v) \\ &= I(j_1, j_2+N_2) - I(j_1, j_2). \end{aligned} \tag{6}$$

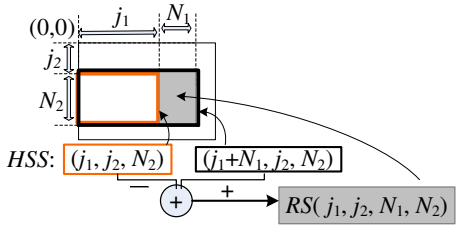


Figure 3. Strip sum and rectangle sum on the image. Only one addition is required to compute $RS(j_1, j_2, N_1, N_2)$ from the data structure $HSS$ using (8).

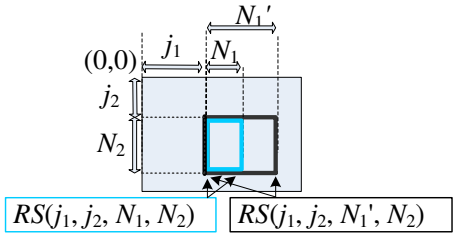


Figure 4. Rectangle sums sharing the same height $N_2$. The two rectangle sums can use the same strip sum for computation.

Table 2. Pseudo-code showing the utilization of strip sum for computing rectangle sums sharing the same height $N_2$.

*Temp*$_{HSS}$ is a 1-D array buffer of size $J_1$.
for $j_2$= 0 to $J_2$ - 1
  Compute the *HSS*($j_1$, $j_2$, $N_2$) for 0≤$j_1$≤$J_1$ at the $j_2$th row from integral image and store them into *Temp*$_{HSS}$.
  for $j_1$=1 to $J_1$
    Compute *RS*($j_1$, $j_2$, $N_1$, $N_2$) using (7) for rectangles having the same $N_2$ but different $N_1$. The *HSS*($j_1$, $j_2$, $N_2$) and *HSS* ($j_1$+$N_1$, $j_2$, $N_2$) in (7) are stored in *Temp*$_{HSS}$.
  end of for $j_1$=1 to $J_1$
end of for $j_2$=1 to $J_2$

Considering $RS(j_1, j_2, N_1, N_2)$ and $RS(j_1, j_2, N_1', N_2)$ $(N_1 \ne N_1')$ for $0 \le j_1, j_1 + N_1, j_1 + N_1' \le J_1$; $0 \le j_2, j_2 + N_2 \le J_2$ , we have:

$$\begin{aligned} &RS(j_1, j_2, N_1, N_2) \\ &= [I(j_1+N_1, j_2+N_2) - I(j_1+N_1, j_2)] \\ &\quad - [I(j_1, j_2+N_2) - I(j_1, j_2)] \\ &= HSS(j_1+N_1, j_2, N_2) - HSS(j_1, j_2, N_2), \\ &RS(j_1, j_2, N_1', N_2) \\ &= HSS(j_1+N_1', j_2, N_2) - HSS(j_1, j_2, N_2). \end{aligned} \tag{7}$$

The computation of $RS(j_1, j_2, N_1, N_2)$ using the strip sum is shown in Fig. 3. As shown in (7), only one addition is required to obtain one rectangle sum using the strip sum defined in (6). The $RS(j_1, j_2, N_1, N_2)$ and $RS(j_1, j_2, N_1', N_2)$ in (7) have the same height $N_2$ but have different width $(N_1 \ne N_1')$. They both use the $HSS(j_1, j_2, N_2)$ for computation. Fig. 4 shows the relationship between $RS(j_1, j_2, N_1, N_2)$ and $RS(j_1, j_2, N_1', N_2)$. According to (7), we require one addition to obtain $RS(j_1, j_2, N_1, N_2)$ for rectangle sums hav-

Table 3. The additions (*adds*) and memory fetch operations (*M-ops*) per pixel required for computing $r$ rectangle sums having $Num_H$ different heights and $Num_W$ different widths.

| | Box [28] | Integral image [38] | Strip sum |
|---|---|---|---|
| *Adds* | 4*r* | 2+3*r* | 2+min{$Num_W$, $Num_H$}+*r* |
| *M-ops* | 6*r* | 4+4*r* | 4+2 min{$Num_W$, $Num_H$}+2*r* |

ing any width $N_1$ and the fixed height $N_2$ at any pixel position using $HSS(j_1, j_2, N_2)$. This procedure of using strip sum for computing rectangle sum is shown in Table 2. This method is also applicable for $45^\circ$ rotated rectangles.

Similarly, in order to use strip sum for rectangle sums having the same width $N_1$, we define the vertical strip sum $VSS(j_1, j_2, N_1)$ as the sum of pixels $X(u, v)$ for $j_1 \le u \le j_1 + N_1 - 1, 0 \le v \le j_2 - 1$. And we can use the same $VSS$ for computing rectangle sums having any height $N_2$ and fixed width $N_1$ as follows:

$$RS(j_1, j_2, N_1, N_2)$$
$$= VSS(j_1, j_2 + N_2, N_1) - VSS(j_1, j_2, N_1),$$
$$\text{where } VSS(j_1, j_2, N_1) = I(j_1 + N_1, j_2) - I(j_1, j_2).$$

### 3.2. Computational complexity analysis

Let $r$ be the number of different sizes for rectangle sums computed on each pixel position. The box technique in [27] requires $4r$ additions and $6r$ memory fetch operations (M-ops) per pixel for computing $r$ rectangle sums. The integral image method in [35] requires $2 + 3r$ additions and $4 + 4r$ M-ops per pixel, where 2 additions and 4 M-ops are used for constructing the integral image, $3r$ additions and $4r$ M-ops are used for computing rectangle sum from integral image.

Let the number of different widths and different heights for $r$ rectangles be $Num_W$ and $Num_H$ respectively. Since we can use either horizontal strip sum or vertical strip sum for computing rectangle sum, we should construct $\min\{Num_W, Num_H\}$ strip sums for these $r$ rectangle sums. The strip sum method requires $2 + \min\{Num_W, Num_H\} + r$ additions and $4 + 2\min\{Num_W, Num_H\} + 2r$ M-ops per pixel for computing $r$ rectangle sums, where 2 additions and 4 M-ops are used for preparing the integral image, $\min\{Num_W, Num_H\}$ additions and $2\min\{Num_W, Num_H\}$ M-ops are used for preparing the strip sums, $r$ additions and $2r$ M-ops are used for obtaining the rectangle sums from strip sums. The computational complexity for different methods is compared in Table 3.

### 3.3. Buffering strip sum

When $r$ rectangle sums having $Num_H$ different heights are computed for $J_1 \times J_2$ image, size $J_1 J_2 Num_H$ memory will be required if the $Num_H$ horizontal strip sums are all stored in memory. Actually, buffering strategy can be used to reduce the memory required.

We use the computation of two rectangle sums having the same height but different width in Fig. 4 as the example for illustration. The procedure of memory usage is described in Table 2. At any row $j_2$, computation of $RS(j_1, j_2, N_1, N_2)$ and $RS(j_1, j_2, N_1', N_2)$ needs only the strip sum at row $j_2$, i.e. $HSS(j_1, j_2, N_2)$ for $0 \le j_1 \le J_1$, but needs not the strip sum at other rows. We can compute the strip sum at row $j_2$ and store the strip sums using a buffer $Temp_{HSS}$ having size $J_1$ but need not store the strip sum at the other rows. Thus size $J_1$ memory is required for storing strip sum in this example.

If $r$ rectangle sums having $Num_H$ different heights are computed, size $J_1 Num_H$ memory are required to store the strip sum data, where $Num_H \le J_2$. Thus the memory required by the strip sum using buffering strategy is smaller than or equal to image size $J_1 J_2$.

### 3.4. Orthogonal Haar transform on sliding windows

Fig. 5 shows the 2-D $4 \times 4$ and $8 \times 8$ OHT we choose for pattern matching. The OHT for other sizes can be similarly derived. Normally, the basis for 2-D image is called basis image. Since 2-D basis image can be represented by 1-D vector, e.g. 2-D candidate window and pattern are represented as 1-D vectors $\vec{X}_w^{(j)}$ and $\vec{X}_p$, basis image is called basis vector in this paper so that the 2-D window represented by 1-D vector is projected onto basis vector instead of basis image. It is easy to see that the OHT basis vectors in Fig. 5 are orthogonal to each other. We can normalize the basis vectors and have orthonormal OHT basis vectors. Thus OHT can be applied for transform domain pattern matching shown in Table 1.

The GCK algorithm requires $2u$ additions for obtaining $u$ GCK projection values. If we directly use the integral image for computation, 7 additions are required to obtain a Haar-like feature and about $7u$ additions are required for obtaining $u$ OHT projection values. For example, the GCK algorithm requires about 32 additions to obtain 16 GCK projection values and the integral image method requires about 112(=$7 \cdot 16$) additions to obtain the 16 projection values of 2-D $4 \times 4$ OHT.

The 2-D $4 \times 4$ OHT in Fig. 5 will be used as an example to illustrate the the computational complexity of the proposed fast algorithm for computing OHT. There are 1 rectangle sum, i.e. basis vector 0, and 4 Haar-like features having different sizes for the 16 basis vectors: basis vector 1 has size $4 \times 4$; basis vectors 2 and 3 have the same size $4 \times 2$ but different positions; basis vectors 4 to 7 have the same size $2 \times 2$ but different positions; basis vectors 8 to 15 have the same size $2 \times 1$ but different positions. Basis vectors having the same size but different positions are the same Haar-like features having different positions on an image. These basis vectors can be obtained from computing one Haar-like feature in a sliding window manner on the im-

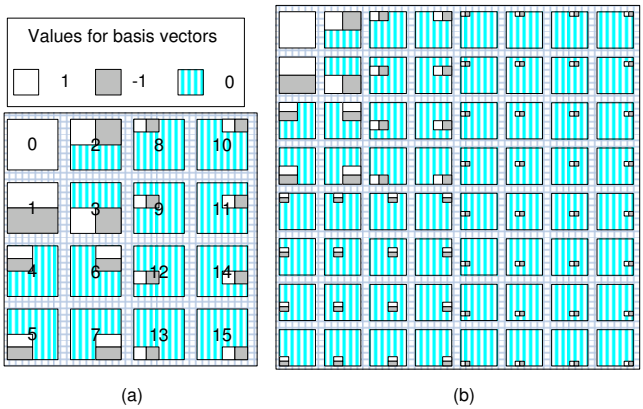


Figure 5. (a): The 2-D $4 \times 4$ OHT basis; (b): the 2-D $8 \times 8$ OHT basis. White represents +1, grey represents $-1$ and green vertical strips represent 0. The numbers for $4 \times 4$ OHT basis denote the order when they are computed in pattern matching.

age. Haar-like features can be obtained from rectangle sums by 1 addition. And the 4 Haar-like features having different sizes can be obtained from the rectangle sums by 4 additions. There are $r = 5$ rectangle sums having different sizes corresponding to the 16 basis vectors: $4 \times 4$, $4 \times 2$ (width 4 and height 2), $2 \times 2$, $2 \times 1$, $1 \times 1$. And we have $Num_H = 3$ different heights: 4, 2 and 1. According the analysis in Table 3, we need $2 + 3 + 5 = 10$ additions for obtaining the 3 strip sums and 5 rectangle sums. In summary, the proposed method requires 14 additions per pixel for computing these 16 Haar-like features, where 4 additions are used for obtaining Haar-like features from rectangle sums and 10 additions are used for obtaining strip sums and rectangle sums.

As another example of the proposed fast algorithm for computing OHT, there are 1 rectangle sum and 6 Haar-like features having different sizes for 2-D $8 \times 8$ OHT in Fig. 5. There are 7 rectangle sums having different sizes corresponding to the 64 basis vectors: $8 \times 8$, $8 \times 4$, $4 \times 4$, $4 \times 2$, $2 \times 2$, $2 \times 1$, $1 \times 1$. And we have $Num_H = 4$ different heights: 8, 4, 2, 1. From the 16 basis vectors of 2-D $4 \times 4$ OHT to the 64 basis vectors of 2-D $8 \times 8$ OHT: the number of Haar-like features and the number of rectangle sums having different sizes increase by 2; $Num_H$ increases by 1.

Generally, when the $N_1 \times N_1$ input data is projected onto the first $u = 4^n, n = 0, 1, ...$ OHT bases, there are 1 rectangle sum and $2 \log_4 u$ Haar-like features having different sizes. There are $r = 2 \log_4 u + 1$ rectangle sums having different sizes: $N_1 \times N_1$, $N_1 \times \frac{N_1}{2}$, $\frac{N_1}{2} \times \frac{N_1}{2}$, $\frac{N_1}{2} \times \frac{N_1}{4}$, $\cdots$, $\frac{N_1}{\sqrt{u}} \times \frac{N_1}{\sqrt{u}}$. And we have $Num_H = log_4 u + 1$. According to the analysis in Table 3, the proposed method requires: 1) $2 + \log_4 u + 1 + 2 \log_4 u + 1$ additions per pixel for obtaining rectangle sums having different sizes; 2) $2 \log_4 u$ additions for obtaining the $2 \log_4 u$ Haar-like features from rectangle sums. In summary, the proposed method needs $4 + 5 \log_4 u$ additions for obtaining $u$ OHT projection values. If the 2-D $N_1 \times N_1$ WHT bases are in the snake order as introduced in [28], then: 1) the first $u$ WHT bases and the first $u$ OHT bases extract the same energy from any input data; 2) the computation for the $u$ WHT bases requires $2u + 2$ additions per window while the computation for the $u$ OHT bases requires $4 + 5 \log_4 u$ additions. For example, when $u = 4, 2u + 2 = 10$ and $4 + 5 \log_4 u = 9$; when $u = 16$, $2u + 2 = 34$ and $4 + 5 \log_4 u = 14$; when $u = 64, 2u + 2 = 130$ and $4 + 5 \log_4 u = 19$.

Considering memory requirement, we need only store the strip sum having one height when computing the basis vectors. For example, when we compute the 0th basis vector of 2-D $4 \times 4$ OHT, we store the horizontal strip sum having height 4. Then we compute the 1st basis vector using the horizontal strip sum having height 2, while the horizontal strip sums having height 4 are not required and need not be stored any more. To further save memory, we can use the buffering strategy in Section 3.3 and require $J_1$ memory for storing strip sum.

Since OHT bases have varying norms, normalization is required. Because the normalization factors are power of 2 for energy, normalization can be computed by shift operations, which has the same cost as addition. This normalization is computed for the partial SSD of candidates, i.e. $||V^{(u \times N)} \vec{X}_t - V^{(u \times N)} \vec{X}_w^{(j)}||^2$ for remaining candidates $\vec{X}_w^{(j)}$, but not for OHT on the entire image, i.e. $V^{(u \times N)} \vec{X}_w^{(j)}$. For $u = 4^n$ OHT bases, there are $2 \log_4 u$ different normalization factors. In worst case, the normalization requires $2 \log_4 u$ shifts per window when no candidate is rejected at all. This normalization procedure requires few computation in practical cases where many candidates are rejected in early stages.

The elements in OHT only contain 1, -1 and 0. The elements in GCK [8] can be real numbers. GCK [8] and generalized GCK [9] are better than OHT in representing a larger set of basis vectors that can be computed efficiently. OHT can be computed in two ways: sliding window or random access. OHT is more efficient than GCK when computed on sliding windows. Another advantage of OHT is in random access. For example, if we want to project 500 candidate windows onto the first basis in a $100 \times 100$ image, OHT can be computed on the 500 candidate windows with the aid of integral image. However, the GCK algorithm has to compute the transformation in a sliding window manner on the entire image such that GCK is computed for about $100^2$ windows instead of for the 500 candidate windows. This kind of situation occurs when only a small number of candidates are remained to be examined in transform domain pattern matching.

## 4. Experimental Results

This section evaluates the performance of pattern matching using OHT by comparing it with FS and the other

Table 4. Datasets and corresponding sizes of images and patterns used in the experiments.

| Dataset | Image Size | Pattern Size |
|---|---|---|
| $S1$ | $160 \times 120$ | $16 \times 16$ |
| $S2$ | $320 \times 240$ | $32 \times 32$ |
| $S3$ | $640 \times 480$ | $64 \times 64$ |
| $S4$ | $1280 \times 960$ | $128 \times 128$ |
| $S5$ | $1280 \times 960$ | $64 \times 64$ |
| $S6$ | $1280 \times 960$ | $32 \times 32$ |

fast FS-equivalent algorithms. All experiments are implemented on a 2.13GHz PC using C on windows XP system with compiling environment VC 6.0. SSD is used for measuring the dissimilarity between pattern and candidate windows. Since all of the algorithms examined find the same result as the FS, we need only compare the computation required by different algorithms.

### 4.1. Dataset and algorithms used for pattern matching experiments

The fast algorithms compared with FS are as follows:
- 1) GCK: the GCK algorithm for WHT in [8];
- 2) WHT: the WHT algorithm in [19];
- 3) IDA: the IDA algorithm in [33];
- 4) OHT$_I$: the integral image method for OHT.
- 5) OHT$_S$: the strip sum method for OHT.

Among the families of the GCK, WHT is the best GCK reported in the literatures because: 1) WHT has good energy packing ability; 2) the computation of WHT requires only 2 additions per projection value using the GCK algorithm. The IDA algorithm is a recently proposed fast FS-equivalent algorithm that uses triangular inequality to formulate lower bound function and prune mismatched windows. GCK, IDA and OHT have similar programming styles. In the experiments, we examine the overall computational efficiency of different algorithms instead of any single step. Thus all preprocessing, e.g. calculating integral image and strip sum, are included for evaluation.

The code for WHT is available online [4] and the code for IDA is provided by the authors of [33]. The parameters for WHT use the default values in [4] and those for IDA are chosen according to [33]. For GCK, we choose the sequency ordered WHT and the code is based on the code used for motion estimation in [28]. As introduced in [19], when the percentage of remaining candidate windows is smaller than certain number $\epsilon$, it is more efficient to directly use SSD for finding the matched windows instead of using transformation on the entire image. In the experiment, $\epsilon$ is set as 2% for WHT, GCK (as recomended in [19, 4]) and set as 0.02% for OHT because 2% is better for WHT and GCK while 0.02% is better for OHT.

Our experiments include a total of 120 images chosen among three databases: MIT [1], medical [3], and remote sensing [2]. The MIT database is mainly concerned with indoor, urban, and natural environments, plus some object categories such as cars and fruits. The two other databases are composed, respectively, of medical (radiographs) and remote sensing (Landsat satellite) images. The MIT dataset is uniformly sampled in a semi-supervised way, meaning that some images are discarded so that the final selected images would belong to all the categories of the dataset (e.g. outdoor natural scenes, urban scenes, indoors, fruits ...). For the remote sensing dataset, the images were selected randomly by hand in order to have different kind of landscapes (e.g. ground, mountains, rivers ...). For the medical image dataset, we selected some images belonging to different human body parts (e.g. arm bones, skull ...).

The 120 images have 4 resolutions which are $160 \times 120$, $320 \times 240$, $640 \times 480$ and $1280 \times 960$. The OpenCV function 'cvResize' with linear interpolation is used for producing the desired resolution of images. Each resolution has 30 images. For each image, 10 patterns were randomly selected among those showing a standard deviation of pixel intensities higher than a threshold (i.e., 45). Six datasets $S1$ to $S6$ with image and pattern sizes given in Table 4 were formed. Each dataset has 300 image-pattern pairs. Datasets $S1$ to $S4$ are the same as those in [33]. Datasets $S5$ and $S6$ are to investigate the effect of pattern size in pattern matching.

In the experiments, if the SSD between a candidate window and the pattern is below a threshold $T$, the candidate window is regarded to match the pattern. For $N_1 \times N_2$ patterns, the threshold $T$ is set as:

$$T = 1.1 \cdot SSD_{min} + N_1 N_2, \tag{8}$$

where $SSD_{min}$ is the SSD between the pattern and the best matching window found by FS.

### 4.2. Pattern matching algorithms on different sizes

In this experiment, we compare the speed-ups yielded by the considered algorithms in pattern matching on datasets $S1$ to $S4$, which correspond to different sizes of image-pattern pairs. The time speed-up or operation speed-up of algorithm $A$ over algorithm $B$ is measured by the execution time or the number of operations required by $B$ divided by that required by $A$. Thus the time speed-up yielded by GCK over FS is the execution time required by FS divided by the time required by GCK in pattern matching for a given dataset. The time speed-ups yielded by GCK, IDA and OHT over FS for the 4 datasets are shown in Fig. 6, where OHT is the fastest. Strip sum requires about 30% to 40% the execution time of integral image in computing OHT. Pattern matching using strip sum for OHT requires about 40% to 60% the time of that using integral image for OHT.

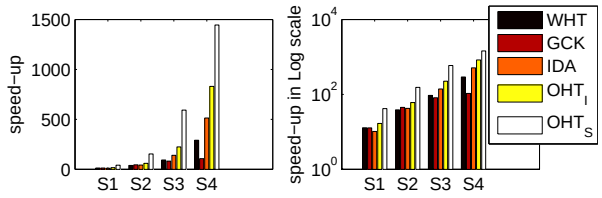


Figure 6. Time speed-up over FS on datasets $S1 - S4$ for different algorithms measured by normal scale (*Left*) and log scale (*right*). The bars for each dataset from left to right correspond to algorithms WHT, GCK, IDA, OHT$_I$ and OHT$_S$.

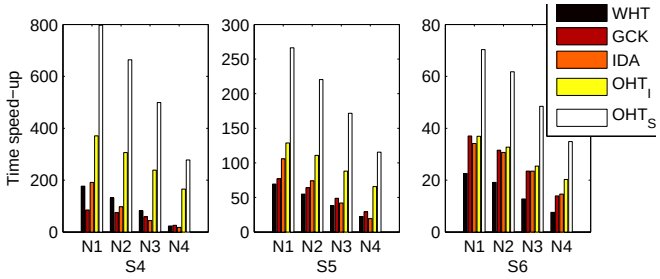


Figure 7. Time speed-up yielded by different algorithms over FS on datasets $S4 - S6$ for noise levels $N1$-$N4$. The bars for each dataset from left to right correspond to algorithms WHT, GCK, IDA, OHT$_I$ and OHT$_S$.

### 4.3. Pattern matching algorithms on different pattern sizes and different noise levels

To evaluate the performance of algorithms on the variation of pattern sizes with image size unchanged, we shall examine the experimental results on datasets $S4$-$S6$. In $S4$-$S6$, the image size is always 1280×960, but the pattern size changes from 128×128 to 32×32. Moreover, 4 different levels of iid zero-mean Gaussian noise were added to each image. The 4 Gaussian noises $N1$, $N2$, $N3$ and $N4$ range from low noise to high noise and have variances being respectively 100, 200, 400 and 800. They correspond to PSNR 28.1, 25.1, 22.1 and 19.2 for the 512×512 image "Lena".

Fig. 7 shows the time speed-ups of the examined fast algorithms over FS in different pattern sizes and different noise levels. It can be seen that OHT is the fastest. Fig. 8 shows the time speed-ups and operation speed-ups of OHT over IDA and GCK on dataset $S4$. As shown in Fig. 8 on dataset $S4$: the time speed-ups of OHT over IDA and GCK are about 4 to 15 and 8 to 10 respectively; the operation speed-ups of OHT over IDA and GCK are about 5 to 24 and 19 to 24 respectively.

## 5. Conclusion

This paper utilizes integral image to obtain the data structure strip sum using one addition per pixel. Then the strip sum is used for computing the sum of pixels within a rectangle, which is called rectangle sum, using one addition independent of the size of the rectangle. The rectangle sum can be regarded as the dc component of the rectangle if we skip the normalization factor. The rectangle sums are building bricks for Haar-like features. We show that strip sum can be used for computing OHT. It is easy to use strip sum for computing the Haar-like features proposed in [32, 36, 22]. Then we propose to use the OHT for FS-equivalent pattern matching. Existing fast algorithms for computing transformation on sliding windows [8, 19, 29] require $O(u)$ additions for projecting input data having size $N$ onto $u$ basis vectors. The OHT can project input data onto $u$ basis vectors by $O(\log u)$ additions using the strip sum. Experimental results show that the proposed algorithm can significantly accelerate the FS-equivalent pattern matching process and outperforms state-of-the-art methods. Strip sum and OHT have potential application in feature extraction, block matching in motion estimation, texture synthesis and analysis, super resolution, image based rendering, image denoising, object detection, tracking, and more.

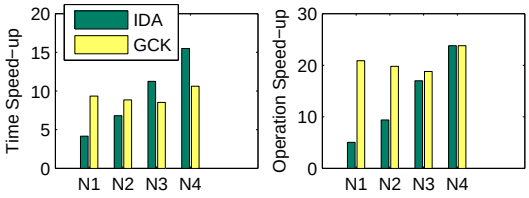


Figure 8. Time speed-up (*left*) and operation speed-up (*right*) of OHT over IDA and GCK at 4 noise levels for dataset $S4$.

## ACKNOWLEDGMENT

The authors wish to thank Professor Yacov Hel-Or at the Interdisciplinary Center and Professor Hagit Hel-Or at the University of Haifa for providing the thesis on generalized GCK and their code implementing GCK and WHT, Dr. Federico Tombari at the University of Bologna for providing their code implementing IDA, image datasets and helpful discussion, Professor Antonio Torralba and CSAIL at the MIT for the use of the MIT database, Professor Rainer Koster and the Institute for Clinical Radiology and Nuclear Medicine of the Lukas Hospital Neuss for the use of the medical image database, and NASA for the use of the remote sensing image database.

## References

[1] ;. http://people.csail.mit.edu/torralba/images.

[2] ;. http://zulu.ssc.nasa.gov/mrsid.

[3] ;. www.data-compression.info/corpora/lukascorpus.

[4] ;. www.faculty.idc.ac.il/toky/software/software.htm.

[5] M. G. Alkhansari. A fast globally optimal algorithm for template matching using low-resolution pruning. *IEEE Trans. Image Process.*, 10(4):526–533, Apr 2001.

[6] Y. Alon, A. Ferencz, and A. Shashua. Off-road path following using region classification and geometric projection constraints. In *CVPR*, volume 1, pages 689–696, Jun. 2006.

[7] H. Bay, T. Tuytelaars, and L. Gool. Surf: Speeded up robust features. In *ECCV*, volume 1, pages 404–417, 2006.

[8] G. Ben-Artz, H. Hel-Or, and Y. Hel-Or. The Gray-code filter kernels. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(3):382–393, Mar. 2007.

[9] G. Ben-Artzi. Gray-code filter kernels (GCK)-fast convolution kernels. Master's thesis, Bar-Ilan Univ., , Ramat-Gan, Israel, 2004.

[10] M. Ben-Yehuda, L. Cadany, and H. Hel-Or. Irregular pattern matching using projections. In *Proc. 12th Int'l Conf. Image Processing (ICIP)*, volume 2, pages 834–837, 2005.

[11] A. Buades, B. Coll, and J.-M. Morel. A non-local algorithm for image denoising. In *CVPR*, volume 2, pages 60– 65, Jun. 2005.

[12] F. Crow. Summed-area tables for texture mapping. In *Proc.11th Ann. Conf. Computer Graphics and Interactive Techniques*, pages 207–212, 1984.

[13] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE Trans. Image Process.*, 16(8):2080–2095, Aug. 2007.

[14] R. M. Dufour, E. L. Miller, and N. P. Galatsanos. Template matching based object recognition with unknown geometric parameters. *IEEE Trans. Image Process.*, 11(12):1385–1396, Dec. 2002.

[15] A. Efros and T. Leung. Texture synthesis by non-parametric sampling. In *ICCV*, pages 1033–1038, Sept. 1999.

[16] A. Fitzgibbon, Y. Wexler, and A. Zisserman. Image-based rendering using image-based priors. In *ICCV*, volume 2, pages 1176–1183, 2003.

[17] W. Freeman, T. Jones, and E. Pasztor. Example-based super-resolution. *IEEE Computer Graphics and Applications*, 22(2):56–65, Mar./Apr 2002.

[18] B. Girod. *Whats Wrong with Mean-Squared Error?*, chapter 15. MIT Press, 1993.

[19] Y. Hel-Or and H. Hel-Or. Real time pattern matching using projection kernels. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(9):1430–1445, Sept. 2005.

[20] Y. Hel-Or, T. Malzbender, and D. Gelb. Synthesis and rendering of 3d textures. In *Texture 2003 Workshop accomp. ICCV 2003,*, pages 53–58, 2003.

[21] C. Lampert, M. Blaschko, and T. Hofmann. Beyond sliding windows: Object localization by efficient subwindow. In *CVPR*, pages 1–8, 2008.

[22] R. Lienhart, A. Kuranov, and V. Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *DAGM 25th Pattern Recognition Symposium*, pages 297–304, 2003.

[23] R. Lienhart and J. Maydt. An extended set of Haar-like features for rapid object detection. In *Proc. IEEE Int. Conf. Image Processing (ICIP)*, volume 1, pages I–900–I–903, Sept. 2002.

[24] T. Luczak and W. Szpankowski. A suboptimal lossy data compression based on approximate pattern matching. *IEEE Trans. Information Theory*, 43(5):1439–1451, 1997.

[25] C. Mak, C. Fong, and W. Cham. Fast motion estimation for H.264/AVC in Walsh Hadamard domain. *IEEE Trans. Circuits Syst. Video Technol.*, 18(5):735–745, Jun. 2008.

[26] S. Mattoccia, F. Tombari, and L. D. Stefano. Fast full-search equivalent template matching by enhanced bounded correlation. *IEEE Trans. Image Process.*, 17(4):528–538, Apr. 2008.

[27] M. J. McDonnell. Box-filtering techniques. *Comput. Graph. Image Process.*, 17:65–70, 1981.

[28] Y. Moshe and H. Hel-Or. Video block motion estimation based on Gray-code kernels. In *IEEE Trans. Image Process.*, volume 18, pages 2243–2254, Oct. 2009.

[29] W. Ouyang and W. Cham. Fast algorithm for Walsh Hadamard transform on sliding windows. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(1):165–171, Jan. 2010.

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

[31] Y. Shina, J. S. Jua, and E. Y. Kim. Welfare interface implementation using multiple facial features tracking for the disabled people. *Pattern Recognition Letters*, 29(13):1784–1796, Oct. 2008.

[32] F. Tang, R. Crabb, and H. Tao. Representing images using nonorthogonal Haar-like bases. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(12):2120–2134, Dec. 2007.

[33] F. Tombari, S. Mattoccia, and L. D. Stefano. Full search-equivalent pattern matching with incremental dissimilarity approximations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(1):129–141, Jan. 2009.

[34] O. Tuzel, F. Porikli, and P. Meer. Pedestrian detection via classification on riemannian manifolds. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(10):1713–1727, Oct. 2008.

[35] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, pages I:511–I:518, 2001.

[36] P. Viola and M. Jones. Robust real-time face detection. *Int'l J. Computer Vision*, 57(2):137–154, 2004.

[37] P. Viola, M. Jones, and D. Snow. Detecting pedestrians using patterns of motion and appearance. In *ICCV*, pages II:734–II:741, 2003.

[38] Q. Wang, J. Mooser, S. You, and U. Neumann. Augmented exhibitions using natural features. *Int'l. J. Virtual Reality*, 7(4):1–8, 2008.

[39] Q. Wang and S. You. Real-time image matching based on multiple view kernel projection. In *CVPR*, 2007.

[40] X. Wu. Template-based action recognition: Classifying hockey players movement. Master's thesis, The University of British Columbia, 2005.

[41] Q. Zhu, M.-C. Yeh, K.-T. Cheng, and S. Avidan. Fast human detection using a cascade of histograms of oriented gradients. In *CVPR 2006*, pages 1491–1498, 2006.