Fast Pattern Matching and its Applications

OUYANG, Wanli

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Doctor of Philosophy
in
Electronic Engineering

The Chinese University of Hong Kong
January 2011

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

題獻/Dedication

獻給我的父母和妻子段立麗

To my parents and my wife Duan Lili

致謝

在博士論文即將完成之際，謹允許我借此向幫助、鼓勵和支持過我的老師、朋友和家人致以深深的敬意和衷心的感謝。

首先要感謝的是我的博士論文導師湛偉權教授。湛教授學識淵博，讓我在自由寬鬆的環境下進行研究。我在攻讀博士學位期間取得的點滴進步都離不開湛教授的悉心指導。而湛教授嚴謹的治學態度和孜孜不倦的工作作風，更是讓我受益終身。

同時要感謝的是圖像與視訊處理實驗室的王曉剛教授、Thierry Blu教授、顏慶義教授和徐孔達教授。他們對各種學術問題的真知灼見以及從不同角度給我提出的建議，同樣給我很大幫助和啟發，他們的建議為我的研究提供了很大幫助。

我也要感謝實驗室的伙伴們。他們是陳麗芝、陳震中、程邦勝、崔春暉、董潔、馮志強、金欣、李超、李宏亮、李傑、李鬆南、劉強、劉雨、麥振文、馬家廉、馬林、潘漢傑、孫德慶、王萌、魏振宇、薛峰、楊文嫻、姚劍、張帆、張茜、張任奇、張偉、趙叢、趙瑞和Y. C. Wong。共同的研究興趣讓我們這些年輕人走到一起，從他們身上我學到了很多。和他們在一起科研和學習，讓我感到充實而快樂，在實驗室度過的這些日子必將成為我非常美好的回憶。

另外，很多國際友人也為我的論文提供了各方面的幫助。Yacov Hel-Or教授和Hagit Hel-Or教授提供了他們關于Generalized GCK論文和他們兩篇論文的源程序。Stefano Mattoccia教授和Federico Tombari博士提供了他們實現IDA算法的源程序和圖像數據，并對我的研究提出了有益的建議。Antonio Torralba和CSAIL提供了MIT圖像數據庫，Rainer Koster和the Institute for Clinical Radiology and Nuclear Medicine of the Lukas Hospital Neuss提供了醫學圖像數據庫。NASA提供了遙感圖像數據庫。

請允許我把最真摯的謝意獻給我的家人，他們是我的堅實後盾。父母數十年的辛勤培養，為我漫長求學生涯提供無限的支持。同時我也要感謝出生不久的兒子，他的出生給我帶來数不清的歡樂。而我的妻子段立麗總是默默地支持我，她温柔的双臂是我人生的避風港，让我能在风雨中栖息又再度遠航。

Acknowledgements

At this moment, I would like to express my gratitude towards professors, colleagues, friends, and family. This thesis would not have been possible without their support.

First and foremost, I would like to thank my thesis advisor Prof. Wai-Kuen Cham for his supervision. Prof. Cham is a learned scholar. He has given me enormous freedom to pursue my own interests. My every little step in research is not achievable without his guidance. His serious attitude on academic research is a benefit for my life and always inspires me to do research work like him. I am indebted to him more than he knows.

Also, it gives me great pleasure to thank Prof. XiaoGang Wang, Prof. Thierry Blu, Prof. King-Ngi Ngan and Prof. Hung-Tat Tsui, who are faculty members of our Lab, for their useful suggestions and illuminations on my research work.

I must surely express my appreciation to my collegues and freinds in IVP lab. They are, Laichi Chan, Zhenzhong Chen, Bangsheng Cheng, Chunhui Cui, Jie Dong, Chi Keung Fong, Xin Jin, Chao Li, Hongliang Li, Jie Li, Songnan Li, Qiang liu, Yu Liu, Chun Man Mak, Kalim Ma, Li ma, Hanjie Pan, Deqing, Sun, Meng Wang, Zhenyu Wei, Feng Xue, Wenxian Yang, Jian Yao, Fan Zhang, Qian Zhang, Renqi Zhang, Wei Zhang, Cong Zhao, Rui Zhao and our lab technician Yuk Chung Wong.

And I wish to acknowledge Prof. Yacov Hel-Or and Prof. Hagit Hel-Or for providing the thesis on generalized GCK and their code implementing GCK and WHT, Prof. Stefano Mattoccia and Dr. Federico Tombari for providing their code implementing IDA, image datasets and helpful discussion, Prof. Antonio Torralba and CSAIL for the use of the MIT database, Prof. 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.

Last but not least, my sincere gratitude goes to my parents for all their tireless efforts in bringing me up and their unlimited love and support. To my lovely new-born

son, thank you for bringing me happiness. Words fail me to express my appreciation to my wife Lili Duan whose dedication, love and persistent support. Her gentle arms are always my safe harbor that comforts me in difficulties and enables me to sail further.

摘要

本論文以提高模板匹配的計算效率為目標。

首先，本文提出一種應用于滑動窗口的沃尔什-哈達瑪變換(WHT)快速算法。該算法可以用于實現快速的模板匹配。

其次，提出對已有等價于全搜索的模板匹配經典算法的分析和比較。受到該分析的啟發，本文提出一組新的變換。該組變換被稱為克羅內克-哈達瑪變換(KHT)。WHT是KHT的成員，基于格雷碼核(GCK)的變換是KHT的子集。因此，KHT提供了更多的表示數據的選擇。然后本文提出KHT算法。該算法比GCK算法更快。所有的KHT都可以使用KHT算法來進行計算。基于KHT，我們找出一組稱為Segmented KHT(SegKHT)的變換。通過將輸入數據分為L_s段，SegKHT需要的計算復雜度是快速KHT算法的$1/L_s$。實驗表明提出的算法明顯加速模板匹配的速度并且比經典算法快。

然后，本論文提出條和(Strip Sum)與正交哈爾變換(Orthogonal Haar Transform)。條和只需要一個加法即可計算一幅圖像任意長方形區域中的像素和。之后，本文提出新的正交哈爾變換。將條和和正交哈爾變換用于模板匹配后，新的快速算法只需要$O(\log u)$個加法即可將大小為$N_1 \times N_2$的二維輸入數據映射到u個正交哈爾變換基。

支撐向量機是一种被被廣泛應用的分類方法。直接計算支撐向量機在需要高運算效率的應用中是不理想的。為了降低運算復雜度，本論文提出具有基于剪枝的變換域的支撐向量機。前面提出的模板匹配快速算法被用于計算支撐向量機。在路人識別的實驗中，該算法具有很高的運算效率。

Abstract

This thesis aims at improving the computational efficiency in pattern matching.

Firstly, this thesis proposes a fast algorithm for Walsh Hadamard Transform (WHT) on sliding windows which can be used to implement pattern matching efficiently.

Then this thesis analyzes and compares state-of-the-art algorithms for full search equivalent pattern matching. Inspired by the analysis, this thesis develops a new family of transforms called the Kronecker-Hadamard Transform (KHT) of which the GCK family is a subset and WHT is a member. Thus, KHT provides more choices of transforms for representing images. Then this thesis proposes a new fast algorithm that is more efficient than the GCK algorithm. All KHTs can be computed efficiently using the fast KHT algorithm. Based on the KHT, this thesis then proposes the segmented KHT (SegKHT). By segmenting input data into L_s parts, the SegKHT requires $1/L_s$ the computation required by the KHT algorithm in computing basis vectors. Experimental results show that the proposed algorithm can significantly accelerate the pattern matching process and outperforms state-of-the-art methods.

After that, strip sum and orthogonal Haar transform are proposed. The sum of pixels in a rectangle can be computed by one addition using the strip sum. Then this thesis proposes to use the orthogonal Haar transform (OHT) for pattern matching. Applied for pattern matching, the fast OHT algorithm using strip sum requires $O(\log u)$ additions per pixel to project input data of size $N_1 \times N_2$ onto u 2-D OHT bases. Experimental results show the efficiency of pattern matching using OHT.

Support vector machine (SVM) is a widely used classification approach. Direct computation of SVM is not desirable in applications requiring computationally efficient classification. To relieve the burden of high computational time required for computing SVM, this thesis proposes a transform domain SVM (TDSVM) using pruning that computes SVM much faster. Experimental results show the efficiency in applying the proposed method for human detection.

Publications

Journal Papers

- **Wanli Ouyang** and Wai Kuen Cham, "Fast algorithm for Walsh Hadamard transform on sliding windows," *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(1):165-171, Jan. 2010.
- **Wanli Ouyang** and Wai-Kuen Cham, "*The Kronecker-Hadamard Transform for Fast Pattern Matching*", Submitted to IEEE Trans. Pattern Anal. Mach. Intell..
- **Wanli Ouyang**, Federico Tombari, Stefano Mattoccia, Luigi Di Stefano, and Wai-Kuen Cham, "*Performance Evaluation of Full Search Equivalent Pattern Matching Algorithms*," IEEE Trans. Pattern Anal. Mach. Intell, Minor revision.

Conference Papers

- **Wanli Ouyang**, Renqi Zhang and Wai-Kuen Cham, "Fast pattern matching using orthogonal Haar transform," In *Proceedings of 2010 IEEE International Conference on Computer vision and pattern recognition (CVPR2010)*, San Fransisco, USA, Jun. 13-18, 2010.

Nomenclature

Abbreviations

1D	One-Dimensional
2D	Two-Dimensional
3D	Three-Dimensional
CC	Cross Correlation
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DTFT	Discrete-Time Fourier Transform
FFT	Fast Fourier Transform
FS	Full Search
GCK	Gray Code Kernel
IDA	Incremental Dissimilarity Approximations
KHT	Kronecker Hadamard Transform
KLT	Karhunen-Loeve Transform
LDA	Linear Discriminant Analysis
LPP	Locality Preserving Projection
LRP	Low Resolution Pruning
NCC	Normalized Cross Correlation
OHT	Orthogonal Haar Transform
PCA	Principal Component Analysis
PSNR	Peak Signal-to-Noise Ratio
SAD	Summation of Absolute Difference
SegKHT	Segmented Kronecker Hadamard Transform
SIFT	Scale Invariant Feature Transform
SSD	Summation of Squared Difference
SVM	Support Vector Machine
WHT	Walsh-Hadamard Transform

Notations

$\mathbf{M}^{(N)}$	WHT matrix of size N
$\mathbf{V}^{(U\times N)}$	Transform matrix of size $U \times N$
$\mathbf{V}_s^{(N)}$	Segmented KHT matrix of size $N \times N$
$\mathbf{I}$	Identity matrix
$\mathbf{S}_l$	Left matrix used in KHT
$\mathbf{S}_r$	Right matrix used in KHT
$\mathbf{V}^T$	Transposition of matrix $\mathbf{V}$
$\vec{\mathbf{m}}$	WHT basis vector
$\vec{\mathbf{x}}^{(N)}$	Input vector of size N
$\vec{\mathbf{x}}_t^{(N)}$	Template represented by vector of size N
$\vec{\mathbf{x}}_w^{(N,j)}$	The jth candidate window represented by vector of size N
$\vec{\mathbf{y}}^{(U)}$	$\vec{\mathbf{y}}^{(U)} = \mathbf{V}^{(U\times N)}\vec{\mathbf{x}}^{(N)}$. Projection value vector of size U
$\vec{\mathbf{y}}^{(N,j)}$	$\vec{\mathbf{y}}^{(U,j)} = \mathbf{V}^{(U\times N)}\vec{\mathbf{x}}_w^{(N,j)}$. Projection value vector at the jth window of size U.
$\vec{\mathbf{v}}^{(N,i)}$	The ith transform basis vector having size N
$\vec{\mathbf{v}}_s^{(N,i)}$	ith SegKHT transform basis vector of size N
$\vec{\mathbf{s}}_l^{(i_l)}$	i_lth basis vector of matrix $\mathbf{S}_l$
$\vec{\mathbf{s}}_r^{(i_r)}$	i_rth basis vector of matrix $\mathbf{S}_r$
$\vec{\mathbf{d}}^{(N/4,j)}$	$\vec{\mathbf{d}}^{(N/4,j)} = \vec{\mathbf{x}}^{(N/4,j)} - \vec{\mathbf{x}}^{(N/4,j+N)}$
$\vec{\mathbf{d}}_{t,w}^{(j)}$	$\vec{\mathbf{d}}_{t,w}^{(j)} = \vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}$
$\vec{\mathbf{z}}$	Any vector
$B(N,P)$	Number of operations required for computing P basis vectors of size N
G_M	$2^{G_M} = N_M$, where N_M is the transform size of WHT matrix
K	Number of input elements, e.g. the image size for 2-D image
L	Size of left matrix $\mathbf{S}_l$
L_p	L_p norm of a vector, e.g. $p = 1$ is absolute sum of elements in the vector
L_s	Size of identity matrix $\mathbf{I}_{L_s}$ used in SegKHT
N	size of input vector $\vec{\mathbf{x}}$
N_M	Transform size of WHT matrix $\mathbf{M}$
N_s	The size of KHT matrix in SegKHT
N_P	The number of partitions used in IDA algorithm
O	Big O notation for computational complexity
P	Partition of given set
R	Size of left matrix $\mathbf{S}_r$
T	Threshold for pattern matching
U	Size of projection value vector $\vec{\mathbf{y}}^{(U)}$
W	Number of candidate windows

$d_N(j)$	$d_N(j) = x_j - x_{j+N}$
f_{low}	Lower bound function
g	Describe how two WHT basis vectors are α-related
h	Small integer used as the parameter in the LRP algorithm
i	Index of basis vector in transform matrix
$t_{N/4}(i,j)$	$t_{N/4}(i,j) = y(N/4,i,j) - y(N/4,i,j+N)$
$y(N,i,j)$	ith projection value at the jth window for window size N
x_j	jth element in vector $\vec{\mathbf{x}}$
$x\%y$	Modulo operation, which is x modulo y
$\mathbf{A} \otimes \mathbf{B}$	Kronecker product of matrix $\mathbf{A}$ and $\mathbf{B}$
$\|x\|$	Absolute value of x
$\|\|\vec{\mathbf{z}}\|\|_p$	L_p norm of vector $\vec{\mathbf{z}}$
$\|\|\mathbf{V}\|\|_p$	Induced L_p norm of matrix $\mathbf{V}$

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Objectives

Pattern matching, also known as template matching, is the task of seeking a given pattern in a given image as illustrated in Figure 1.1. The pattern matching task can be regarded as a degenerated pattern recognition problem where we classify a candidate window as a non-pattern class or a pattern class.

Pattern matching is widely adopted for tasks such as industrial inspection (quality control, defect detection) and fiducial-based pick-and-place. It is also used in signal processing, computer vision, image and video processing. It has found applications in manufacturing for quality control [3], image based rendering [4], image compression [5], object detection [6], super resolution [7], texture synthesis [8], block matching in motion estimation [2; 9], image denoising [10–12], road/path tracking [13], mouth tracking [14], image matching [15], image alignment [16] and action recognition [17]. In many such applications, however, a main problem is the high computational time required by pattern matching. This problem is even more severe when dealing with video data in which pattern matching is done for a large amount of images. This thesis aims at dealing with this problem and proposes many fast pattern matching algorithms.

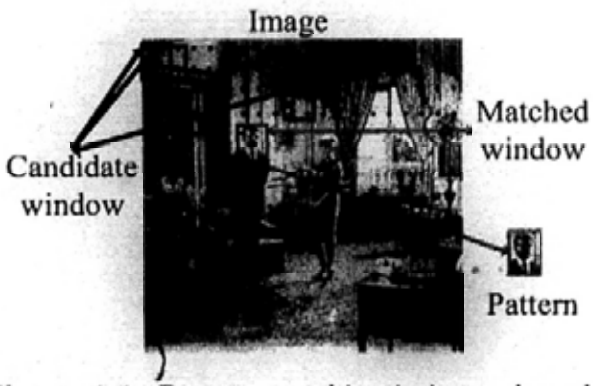


Figure 1.1: *Pattern matching in image 'couple'.*

1.1.1 Measures Used in Pattern Matching

Suppose an $N_1 \times N_2$ pattern has to be sought in a given $J_1 \times J_2$ image of $J = J_1 J_2$ pixels as shown in Figure 1.1. The pattern will be compared with candidate windows of similar size in the image. The Full Search (FS) algorithm computes a similarity or dissimilarity measure between the *pattern* and all its equally-sized *candidate windows* that can be extracted out of the image. We represent the pattern (template) as a length-N vector $\vec{\mathbf{x}}_t^{(N)}$ and the candidate windows as $\vec{\mathbf{x}}_w^{(N,j)}$, 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 16×16 pattern is searched in a 256×256 image, we have $N = 256$, $J = 65536$ and $W = (256 - 16 + 1)^2 = 58081$.

There are several ways to evaluate the similarity or dissimilarity between the pattern and the candidate window. These are represented by different matching measures that can be used for the comparison. In particular, we denote as $d(\vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{x}}_w^{(N,j)})$ the distance between $\vec{\mathbf{x}}_t^{(N)}$ and $\vec{\mathbf{x}}_w^{(N,j)}$, which measures the dissimilarity between $\vec{\mathbf{x}}_t^{(N)}$ and $\vec{\mathbf{x}}_w^{(N,j)}$. The smaller is $d(\vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{x}}_w^{(N,j)})$, the more similar are $\vec{\mathbf{x}}_t^{(N)}$ and $\vec{\mathbf{x}}_w^{(N,j)}$. There are two different situations in pattern matching: 1) detect all candidate windows having $d(\vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{x}}_w^{(N,j)}) < T$, for a given threshold T; 2) find the window that leads to the minimum value of $d(\vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{x}}_w^{(N,j)})$ among all candidate windows. In this thesis, we consider mainly situation 1, where $\vec{\mathbf{x}}_w^{(N,j)}$ is said to match $\vec{\mathbf{x}}_t^{(N)}$ when $d(\vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{x}}_w^{(N,j)}) < T$. As illustrated later, the algorithms for situation 1 can be easily modified to deal with situation 2.

There are researches using similarity or dissimilarity measures that make pattern matching robust to rotation, affine transformations, occlusion and illumination variations. For example, the SSD between the pattern descriptors and the candidate window descriptors is used as dissimilarity measure in [18–22]; Hamming distance is used as the dissimilarity measure in [23]; normalized cross correlation (NCC) is used as similarity measure in [24–26].

In this thesis, we will consider a class of matching measures that find a vast use in template matching applications, i.e. those derived from a distance measure based on the L_p norm. The L_p norm of length-N vector $\vec{\mathbf{z}} = [z_0, z_1, \cdots, z_{N-1}]^T$ is defined as:

$$||\vec{\mathbf{z}}||_p = (|z_0|^p + |z_1|^p + \cdots + |z_{N-1}|^p)^{1/p}. \tag{1.1}$$

Based on the L_p norm, the dissimilarity between $\vec{\mathbf{x}}_t^{(N)}$ and $\vec{\mathbf{x}}_w^{(N,j)}$ can be measured as $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_p^p$. If $p = 1$, then the distance is the sum of absolute differences (SAD), while $p = 2$ yields the sum of squared differences (SSD). As pointed out in [1], though there are arguments against the SSD as a dissimilarity measure for images, it is still widely adopted due to its simplicity. Discussions concerning the use of the SSD as a similarity metric can be found in [27–29]. On the other hand, SAD is computationally more efficient than SSD. In the followings, we introduce some existing algorithms that use SSD and SAD as the dissimilarity measure.

1.2 Fast Pattern Matching Algorithms

Since the FS algorithm is unacceptably slow in most applications, many faster approaches have been proposed in literature [1; 26; 30–45]. The algorithms can be grouped into full search nonequivalent algorithms and full search equivalent algorithms. *Full search nonequivalent* algorithms yield computational savings by reducing the search space [36–38] or by approximating patterns and windows using polynomials [39–41] or linear combination of simple features [42].

Conversely, exhaustive (or *FS-equivalent*) algorithms accelerate the pattern matching process and, at the same time, yield exactly the same result as the FS. In the case of a dissimilarity-based search, a simple approach is known as Partial Distortion Elimination (PDE) [43] and its high efficiency consists in terminating the evaluation of the current dissimilarity measure as soon as it rises above the current minimum. Another approach suitable to dissimilarity-based searches consists in defining a rapidly computable lower bounding function of the adopted distortion measure, so as to check quickly one or more sufficient conditions to skip mismatched positions without carrying out the heavier computations required by the evaluation of the actual dissimilarity measure. Examples of such an approach include the algorithms in [1; 30–33; 44; 45].

1.2.1 Full search approach

With the FS approach, the distance between the pattern $\vec{\mathbf{x}}_t^{(N)}$ and each candidate window $\vec{\mathbf{x}}_w^{(N,j)}$, i.e. $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_p^p$ for $j = 0, \ldots, W-1$, is measured. If $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_p^p < T$, then the window $\vec{\mathbf{x}}_w^{(N,j)}$ is considered as a matching window, otherwise, it is considered as a mismatched window.

When SSD is used, the FS requires about $2NW$ additions and NW multiplications. When SAD is used, the FS requires about $2NW$ additions and NW absolute value operations.

1.2.2 Fast Fourier Transform

The FFT-based approach can be used only with the SSD. As pointed out in [31], the SSD function can be written as

$$||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_2^2 = ||\vec{\mathbf{x}}_t^{(N)}||_2^2 + ||\vec{\mathbf{x}}_w^{(N,j)}||_2^2 - 2 \cdot <\vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{x}}_w^{(N,j)}>, \tag{1.2}$$

where $< \vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{x}}_w^{(N,j)} >= (\vec{\mathbf{x}}_t^{(N)})^T \vec{\mathbf{x}}_w^{(N,j)}$ is the inner product between $\vec{\mathbf{x}}_t^{(N)}$ and $\vec{\mathbf{x}}_w^{(N,j)}$. The FFT facilitates efficient computation of the inner product $< \vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{x}}_w^{(N,j)} >$ in (1.2).

As pointed out in [46], the FFT-based approach requires about $6Jlog_2J$ additions and $6Jlog_2J$ multiplications for computing the inner product. After that, W additions are required to obtain $2 < \vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{x}}_w^{(N,j)} >$ from $< \vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{x}}_w^{(N,j)} >$ for $j = 0, 1, \ldots, W-1$. To compute the term $||\vec{\mathbf{x}}_w^{(N,j)}||_2^2$ at each pixel location, J multiplications are required for squaring pixel values, $4W$ and $5W$ additions are required respectively by the box-filtering technique [47] and the integral image approach [48] for summing up the squared values $||\vec{\mathbf{x}}_w^{(N,j)}||_2^2 = \sum_{x_{w,j,a} \in \vec{\mathbf{x}}_w^{(N,j)}} (x_{w,j,a})^2$ for $j = 0, \cdots, W-1$. The amount of computation needed to obtain $||\vec{\mathbf{x}}_t^{(N)}||_2^2$ is negligible. Finally, $2W$ additions are needed for summing up the three terms in (1.2). In summary, the FFT based approach requires at least $6Jlog_2J + 7W$ additions and $6Jlog_2J + J$ multiplications.

1.2.3 Incremental Dissimilarity Approximations Algorithm

The Incremental Dissimilarity Approximations (IDA) technique relies on partitioning the pattern vector, $\vec{\mathbf{x}}_t^{(N)}$, and each candidate vector, $\vec{\mathbf{x}}_w^{(N,j)}$, into a certain number of sub-vectors in order to determine a succession of pruning conditions characterized by increasing tightness and computational weight. Given an N-dimensional vector, IDA establishes a partition, P, of the vector into N_P disjoint sub-vectors (not necessarily with the same number of components). In particular, it defines a partition of set $\{1, 2, \ldots N\}$ into N_P disjoint sub-sets $P_1, \cdots, P_{N_P}$, where $P = \{P_1, P_2, \ldots, P_{N_P}\}$, $\cup_{m=1}^{N_P} P_m = \{1, 2, \ldots N\}$, $P_a \cap P_b = \phi, \forall a \neq b, a, b \in 1, 2, \cdots, N_P$, $1 \leq N_P \leq N$.

Given P, IDA defines the *partial* L_p-norm of vector $\vec{\mathbf{x}}_t^{(N)}$ and $\vec{\mathbf{x}}_w^{(N,j)}$ limited to the

sub-vector associated with $P_m \in P$ as:

$$||\vec{\mathbf{x}}_t^{(N)}||_{p,P_m} = \Big(\sum_{n \in P_m} |x_{t,n}|^p \Big)^{\frac{1}{p}}, \tag{1.3}$$

$$||\vec{\mathbf{x}}_w^{(N,j)}||_{p,P_m} = \Big(\sum_{n \in P_m} |x_{w,j,n}|^p \Big)^{\frac{1}{p}}, \tag{1.4}$$

where $x_{t,n}$ is the nth element in pattern $\vec{\mathbf{x}}_t^{(N)}$ and $x_{w,j,n}$ is the nth element in window $\vec{\mathbf{x}}_w^{(N,j)}$. In addition, IDA defines the *partial* L_p-dissimilarity between $\vec{\mathbf{x}}_t^{(N)}$ and $\vec{\mathbf{x}}_w^{(N,j)}$ limited to the sub-vector associated with $P_m \in P$ as:

$$||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_{p,P_m}^p = \sum_{n \in P_m} |x_{t,n} - x_{w,j,n}|^p. \tag{1.5}$$

Then, by virtue of the *triangular inequality* applied on corresponding sub-vectors IDA establishes the following N_P inequalities:

$$||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_{p,P_m}^p \geq \big| ||\vec{\mathbf{x}}_t^{(N)}||_{p,P_m} - ||\vec{\mathbf{x}}_w^{(N,j)}||_{p,P_m} \big|^p, m = 1, .., N_P \tag{1.6}$$

and summing up both members of the inequalities attains a *lower bound* of the function measuring the dissimilarity between $\vec{\mathbf{x}}_t^{(N)}$ and $\vec{\mathbf{x}}_w^{(N,j)}$:

$$||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_p^p \geq \sum_{t=1}^{N_P} \big| ||\vec{\mathbf{x}}_t^{(N)}||_{p,P_m} - ||\vec{\mathbf{x}}_w^{(N,j)}||_{p,P_m} \big|^p. \tag{1.7}$$

This inequality provides a sufficient condition that allows for pruning those candidates which cannot represent a matching position. In fact, if the lower-bound of the dissimilarity function exceeds the threshold T that discriminates between matching and mismatching candidates:

$$\sum_{m=1}^{N_P} \big| ||\vec{\mathbf{x}}_t^{(N)}||_{p,P_m} - ||\vec{\mathbf{x}}_w^{(N,j)}||_{p,P_m} \big|^p > T \tag{1.8}$$

then, from (1.7) and (1.8), $\vec{\mathbf{x}}_w^{(N,j)}$ cannot be a matching pattern.

Moreover, should (1.8) not be satisfied, rather than computing from scratch the term $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_p^p$, IDA obtains another pruning condition based on a tighter lower-bound by considering a sub-vectors pair and replacing in the left-hand term of (1.8) the difference between the *partial* L_p-norms with the corresponding *partial* L_p-dissimilarity.

This process can be iteratively applied to all the N_P sub-vectors pairs resulting from

P, so as to determine up to N_P sufficient conditions that can be sequentially checked when matching each candidate vector $\vec{\mathbf{x}}_w^{(N,j)}$. In particular, the tightness of the lower-bounding function can be further increased by taking the next sub-vectors pair and, again, replacing the difference between the *partial* L_p-norms with the corresponding *partial* L_p-dissimilarity. These N_P conditions are based on the following succession of increasingly tighter lower-bounds:

$$\begin{aligned}
&\sum_{m=1}^{N_P} \left| ||\vec{\mathbf{x}}_t^{(N)}||_{p,P_m} - ||\vec{\mathbf{x}}_w^{(N,j)}||_{p,P_m} \right|^p \leq \\
\leq ||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_{p,P_i}^p &+ \sum_{m=1,m\neq i}^{N_P} \left| ||\vec{\mathbf{x}}_t^{(N)}||_{p,P_m} - ||\vec{\mathbf{x}}_w^{(N,j)}||_{p,P_m} \right|^p \leq \\
&\leq ||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_{p,P_i}^p + ||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_{p,P_k}^p + \\
&+ \sum_{m=1,m\neq i,k}^{N_P} \left| ||\vec{\mathbf{x}}_t^{(N)}||_{p,P_m} - ||\vec{\mathbf{x}}_w^{(N,j)}||_{p,P_m} \right|^p \leq \cdots \\
\cdots \leq \sum_{m=1,m\neq N_P}^{N_P} ||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_{p,P_m}^p &+ \left| ||\vec{\mathbf{x}}_t^{(N)}||_{p,P_{N_P}} - ||\vec{\mathbf{x}}_w^{(N,j)}||_{p,P_{N_P}} \right|^p. \qquad (1.9)
\end{aligned}$$

Hence, throughout the matching process, each vector $\vec{\mathbf{x}}_w^{(N,j)}$ undergoes checking a succession of sufficient conditions. Should the last condition be not verified, the process ends up in computing the dissimilarity $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_p^p$ simply by replacing the difference between the *partial* L_p-norms with the corresponding *partial* L_p-dissimilarity of the last sub-vectors pair.

1.2.4 Transform Domain Pattern Matching

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

$$\vec{\mathbf{y}}^{(U)} = \mathbf{V}^{(U\times N)}\vec{\mathbf{x}}^{(N)} = [\vec{\mathbf{v}}^{(N,0)} \ \ldots \ \vec{\mathbf{v}}^{(N,U-1)}]^T\vec{\mathbf{x}}^{(N)}, \qquad (1.10)$$

where $.^T$ is matrix transposition, vector $\vec{\mathbf{x}}^{(N)}$ of length N is called input window, vector $\vec{\mathbf{y}}^{(U)}$ of length U is called projection value vector, the U elements in vector $\vec{\mathbf{y}}^{(U)}$ are called projection values and $\mathbf{V}^{(U\times N)}$ is a $U \times N$ matrix that contains U orthogonal basis vectors $\vec{\mathbf{v}}^{(N,i)}$ of length N for $i = 0, \ldots, U-1$. When $\mathbf{V}^{(U\times N)}$ is a square matrix, i.e. $U = N$, we denote it as $\mathbf{V}^{(N)}$. The transformation is called projection in [1; 35]. Basis vector is called projection kernel in [1] and called filter kernel in [32].

The following inequality is proved in [1] when the u basis vectors in $\mathbf{V}^{(u\times N)}$ are

orthonormal:

$$\begin{aligned} &||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||^2 \geq ||\mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_t^{(N)} - \mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_w^{(N,j)}||^2 = ||\vec{\mathbf{y}}_t^{(u)} - \vec{\mathbf{y}}_w^{(u,j)}||^2, \\ &\text{where } \vec{\mathbf{y}}_t^{(u)} = \mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_t^{(N)}, \vec{\mathbf{y}}_w^{(u,j)} = \mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_w^{(N,j)}. \end{aligned} \tag{1.11}$$

If $||\vec{\mathbf{y}}_t^{(u)} - \vec{\mathbf{y}}_w^{(u,j)}||^2 > T$, then we have $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||^2 > T$ from (1.11), and so we can safely prune candidate window $\vec{\mathbf{x}}_w^{(N,j)}$ from set_{can}. In this way, $||\vec{\mathbf{y}}_t^{(u)} - \vec{\mathbf{y}}_w^{(u,j)}||^2 > T$ is a sufficient rejection condition for rejecting mismatched window. Denote the set of candidates as set_{can}, which initially contains all candidates. For each iteration of u, where u increases from 1, $\vec{\mathbf{x}}_t^{(N)}$ and $\vec{\mathbf{x}}_w^{(N,j)}$ are projected onto transform domain using $\mathbf{V}^{(u\times N)}$ and the rejection condition is checked for the remaining candidates in set_{can}. Finally, after N_{Maxu} number of such iteration, the remaining candidate windows in set_{can} undergo FS for finding out the matched windows. This procedure is summarized in Table 1.1. Such pattern matching approach is FS equivalent. The u basis vectors in $\mathbf{V}^{(u\times N)}$ are selected from the U orthonormal vectors $\vec{\mathbf{v}}^{(N,0)}$ ... $\vec{\mathbf{v}}^{(N,U-1)}$ in $\mathbf{V}^{(U\times N)}$.

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

Table 1.1: *Transform domain pattern matching*

The main advantage of using transformation is that it is more efficient computing $||\mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_t^{(N)} - \mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_w^{(N,j)}||^2$ than computing $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||^2$ and a small number of projection values can eliminate a large number of mismatched windows. According to [1], pattern matching using Walsh Hadamard Transform (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, Ben-Yehuda et al. helps Gray-Code Kernels (GCK) and WHT to deal with this problem by segmenting the pattern into dyadic components in [35]. Because of these advantages, transform domain pattern matching has found application in block matching in motion estimation for video coding [2; 9], tracking [13; 14], feature point based image matching [15], texture synthesis [49] and augmented reality [50].

As analyzed in [1], the computational efficiency of transform domain pattern matching is dependent on two factors: 1) the cost of computing transformation; 2) the ability of the rejection condition $||\vec{y}_t^{(u)} - \vec{y}_w^{(u,j)}||^2 > T$ in rejecting mismatched windows and thus saving the computation required afterwards, which is determined by the energy packing ability of transformation. The energy packing ability corresponds to the ability of a transformation in compacting energy from the input data into as small number of projection values as possible, i.e. the ability in using small u to obtain large $||\vec{y}_t^{(u)} - \vec{y}_w^{(u,j)}||^2$. The larger is $||\vec{y}_t^{(u)} - \vec{y}_w^{(u,j)}||^2$, the more powerful is the rejection condition $||\vec{y}_t^{(u)} - \vec{y}_w^{(u,j)}||^2 > T$ in pruning mismatched windows. In summary, the transformation should be computationally efficient in packing energy.

Hel-Or and Hel-Or find in [1] that WHT is efficient for transform domain pattern matching. Their algorithm requires $2N-2$ additions for obtaining all WHT projection values in each window of size N. Following this work, a number of algorithms have been proposed for computing WHT. The GCK algorithm proposed in [32] requires a similar number of additions as [1] when all projection values are computed and requires fewer number of additions when only a small number of projection values are computed.

Meanwhile, new families of transforms that can be efficiently computed by fast algorithms are also proposed. The GCK [32], which has WHT on sliding windows as a member, is a family of transforms that can be computed efficiently by the GCK algorithm. The generalized GCK [51], which is a superset of GCK, can be computed efficiently by the approach in [51]. Among the families of the GCK and the generalized GCK, the most efficient transform domain pattern matching uses WHT. However, the transformation takes the main computational time in pattern matching. Thus the high computational requirement of transformation is still the bottleneck of efficient transform domain pattern matching. Therefore, it is desirable to to perform the transformation more efficiently.

1.3 Thesis Outline

This thesis focuses on fast pattern matching algorithms. Human detection will be used as the application of the proposed algorithm.

In Chapter 2, this thesis proposes a fast algorithm for WHT on sliding windows which can be used to implement pattern matching efficiently. The computational requirement of the proposed algorithm is about 1.5 additions per projection value per sample.

After developing the fast algorithm in Chapter 2, this thesis proposes an analysis and comparison of state-of-the-art algorithms for full search equivalent pattern matching in Chapter 3. Our intention is that the datasets and tests used in our evaluation will be a benchmark for testing future pattern matching algorithms, and that the analysis concerning the state-of-the-art algorithms could inspire new fast algorithms.

The GCK family which has WHT on sliding windows as a member is a family of kernels that can perform image analysis efficiently using a fast algorithm such as the GCK algorithm. The GCK has been successfully used for pattern matching. In Chapter 4, this thesis develops a new family of transforms called the Kronecker-Hadamard Transform (KHT) of which the GCK family is a subset. Thus, KHT provides more choices of transforms for representing data. Then this thesis proposes a new fast algorithm that is more efficient than the GCK algorithm. The proposed fast KHT algorithm requires 4 additions per pixel for computing 3 basis vectors independent of transform size and dimension. All KHTs can be computed efficiently using the fast KHT algorithm. Based on the KHT, this thesis then proposes the segmented KHT (SegKHT). By segmenting input data into L_s parts, the SegKHT requires only 4 additions per pixel for computing $3L_s$ basis vectors. Experimental results show that the proposed algorithm can significantly accelerate the full-search equivalent pattern matching process and outperforms state-of-the-art methods.

Chapter 5 proposes strip sum on the image. The sum of pixels in a rectangle can be computed by one addition using the strip sum. Then this thesis proposes to use the orthogonal Haar transform (OHT) for pattern matching in Chapter 5. Applied for pattern matching, the algorithm using strip sum requires $O(\log u)$ additions per pixel to project input data of size $N_1 \times N_2$ onto u 2-D OHT basis. Experimental results show the efficiency of full search equivalent pattern matching using OHT.

Support vector machines are widely used for classification applications. Direct computation of support vector machines (SVMs) is not desirable in applications requiring computationally efficient classification. To relieve the burden of high computational time required for computing SVM, this thesis proposes a transform domain SVM (TDSVM) that computes SVM much faster in Chapter 6. Experimental results show the efficiency in applying the proposed method for human detection.

Finally, in Chapter 7, the contributions of this thesis are summarized and the future research directions are discussed.

Chapter 2

Fast Algorithm for Walsh Hadamard Transform on Sliding Windows

2.1 Introduction

Pattern matching, also named as template matching, is the procedure of seeking a given pattern in a given signal data. For pattern matching in images, the pattern is usually a 2D image fragment which is much smaller than the image. It has been found that pattern matching can be performed efficiently in Walsh Hadamard Transform (WHT) domain [1]. In pattern matching, signal vectors obtained by a sliding window need to be compared to a sought pattern. Hel-Or and Hel-Or's algorithm [1] requires $2N - 2$ additions for obtaining all WHT projection values in each window of size N. Note that one subtraction is considered to be one addition regarding the computational complexity in this chapter. Their algorithm achieves efficiency by utilizing previously computed values in the internal vertices of the tree structure in Fig. 2.1. Recently, the Gray Code Kernel (GCK) algorithm [32] which utilizes previously computed values in the leaves of the tree structure in Fig. 2.1 was proposed. The GCK algorithm requires similar computation as [1] when all projection values are computed and requires less computation when only a small number of basis vectors are computed.

In this chapter, we propose a fast algorithm for WHT on sliding windows. Instead of performing order-N WHT by means of order-$N/2$ WHT and N additions in the tree structure, which is the technique adopted in [1], the proposed algorithm computes order-N WHT by means of order-$N/4$ WHT and $N + 1$ additions. In this way, the proposed algorithm can obtain all WHT projection values using about $3N/2$ additions per window. In the computation of partial projection values for sliding windows, the proposed algorithm requires only 1.5 additions per basis vector for each window. As shown by experimental results in Section 2.6, the computational time required by the

proposed algorithm for computing 10 or more projection values is about 75 percent of that of the GCK algorithm.

The rest of the chapter is organized as follows: Section 2.2 defines terms and symbols used in this chapter. Then, the WHT algorithm in [1] is briefly introduced. In Section 2.3, we introduce two examples of the proposed algorithm. Section 2.4 illustrates the proposed algorithm for 1D order-N WHT. The algorithm computes order-N WHT using order-4 and order-$N/4$ WHT. In Section 2.5, the number of additions required by the proposed algorithm is derived. Section 2.6 gives the experimental result of motion estimation in video coding application, which utilizes the proposed algorithm for computing 2D WHT on sliding windows. Finally, Section 2.7 presents summary.

2.2 Walsh Hadamard Transform on sliding windows

2.2.1 Definitions

Consider J input signal elements x_n, where $n = 0, 1, \ldots, J-1$, which will be divided into overlapping windows of size N ($J > N$). Let the jth input window be:

$$\vec{\mathbf{x}}^{(N,j)} = [x_j, x_{j+1}, \ldots, x_{j+N-1}]^T, \text{ for } j = 0, 1, \ldots, J-N \tag{2.1}$$

A 1D order-N WHT transforms N numbers into N projection values. Let $\mathbf{M}^{(N)}$ be an order-N WHT matrix and

$$\mathbf{M}^{(N)} = [\vec{\mathbf{m}}^{(N,0)}, \ldots \vec{\mathbf{m}}^{(N,i)}, \ldots \vec{\mathbf{m}}^{(N,N-1)}]^T, \tag{2.2}$$

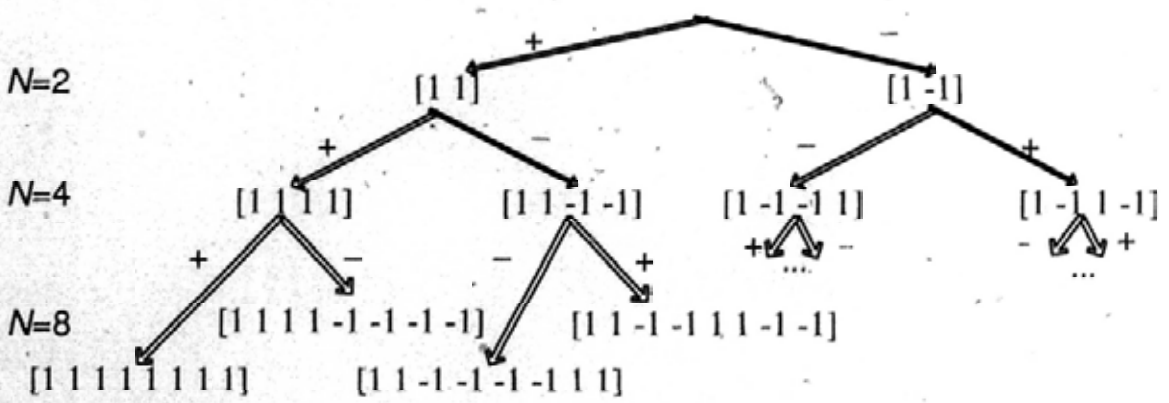


Figure 2.1: *Tree structure for Walsh-Hadamard Transform in sequency order.*

where $\mathbf{M}^{(N)}$ is an $N \times N$ matrix , $\vec{\mathbf{m}}^{(N,i)}$ for $i = 0, \ldots, N-1$ is the ith WHT basis vector having length N and

$$N = 2^{G_M}, G_M = 0, 1, 2, \ldots. \tag{2.3}$$

Let $y(N, i, j)$ for $i = 0, 1, \ldots, N-1; j = 0, 1, \ldots, J-N$ be the ith WHT projection value for the jth window and

$$y(N, i, j) =< \vec{\mathbf{m}}^{(N,i)}, \vec{\mathbf{x}}^{(N,j)} >= \vec{\mathbf{m}}^{(N,i)^T} \vec{\mathbf{x}}^{(N,j)}. \tag{2.4}$$

In [32], $\vec{\mathbf{m}}^{(N,i)}$ and $y(N, i, j)$ are called the ith projection kernel and projection result, respectively.

Let $\vec{\mathbf{y}}^{(N,j)}$ be the projection value vector containing all order-N WHT projection values at the jth window and

$$\vec{\mathbf{y}}^{(N,j)} = \begin{bmatrix} y(N,0,j) \\ y(N,1,j) \\ \vdots \\ y(N,N-1,j) \end{bmatrix} = \mathbf{M}^{(N)} \vec{\mathbf{x}}^{(N,j)} = \begin{bmatrix} \vec{\mathbf{m}}^{(N,0)^T} \\ \vec{\mathbf{m}}^{(N,1)^T} \\ \vdots \\ \vec{\mathbf{m}}^{(N,N-1)^T} \end{bmatrix} \vec{\mathbf{x}}^{(N,j)}. \tag{2.5}$$

For example, when $N = 4$, we have

$$\vec{\mathbf{y}}^{(4,j)} = \begin{bmatrix} y(4,0,j) \\ y(4,1,j) \\ y(4,2,j) \\ y(4,3,j) \end{bmatrix} = \mathbf{M}^{(4)} \vec{\mathbf{x}}^{(N,j)} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} x_j \\ x_{j+1} \\ x_{j+2} \\ x_{j+3} \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{m}}^{(4,0)^T} \\ \vec{\mathbf{m}}^{(4,1)^T} \\ \vec{\mathbf{m}}^{(4,2)^T} \\ \vec{\mathbf{m}}^{(4,3)^T} \end{bmatrix} \vec{\mathbf{x}}^{(N,j)}. \tag{2.6}$$

2.2.2 Previous WHT Computation Methods

In [1], Hel-Or and Hel-Or proposed a fast algorithm that computes $\vec{\mathbf{y}}^{(N,j)}$ from $\vec{\mathbf{y}}^{(N/2,j)}$ and $\vec{\mathbf{y}}^{(N/2,j+N/2)}$ using (2.7):

$$y(N, i, j) = \begin{cases} y(N/2, \lfloor i/2 \rfloor, j) + y(N/2, \lfloor i/2 \rfloor, j + N/2), & i\%4 = 0 \text{ or } 3, \\ y(N/2, \lfloor i/2 \rfloor, j) - y(N/2, \lfloor i/2 \rfloor, j + N/2), & i\%4 = 1 \text{ or } 2, \end{cases} \tag{2.7}$$

where % is the modulo operation and $\lfloor \cdot \rfloor$ is the floor function.

As shown in Fig. 2.1, their algorithm first computes WHT projection values for

window size N being 2, which are then used to compute WHT projection values for window size being 4, and so on. The computation starts at the root and moves down the tree until the projection values represented by the leaves are computed using (2.7). The algorithm in [1] requires one addition per window along each node of the tree in Fig. 2.1. The GCK algorithm [32] utilizes previously computed order-N projection values for computing the current order-N projection value. When a small number of projection values are computed, the GCK algorithm requires two additions per window for each projection value, while the algorithm in [32] requires $O(\log N)$ additions. When all projection values are computed, both the algorithms in [1] and [32] require about $2N$ additions.

A new fast algorithm, which is more efficient than those reported in [1] and [32], is proposed in this chapter. It can efficiently compute order-N WHT on sliding windows, i.e., $y(N, i, j)$ for $i = 0, 1, \ldots, P-1 (P \leq N)$ and $j = 0, 1, 2, \ldots J - N$.

2.3 Fast Algorithm for WHT on Sliding Windows for window Sizes 4 and 8

This section gives examples of computing order-N WHT on sliding windows of sizes $N = 4$ and 8 using the proposed algorithm.

2.3.1 Fast Algorithm for Window Size 4

The proposed algorithm and the GCK algorithm [32] for window size being 4 are described in Table 2.1. The proposed algorithm computes $\vec{\mathbf{y}}^{(4,j+1)}$ (the WHT projection values in window $j+1$) using $\vec{\mathbf{y}}^{(4,j)}$ (the computed projection values in window j), as shown in Table 2.1. Except for the 0th projection value $y(4, 0, j+1)$, the GCK algorithm utilizes the previous order-4 projection values to compute the current order-4 projection value.

Define $d_N(j)$ as:

$$d_N(j) = x_j - x_{j+N}. \tag{2.8}$$

We can see from Table 2.1 that the WHT projection values in window $j+1$ can be

	x_j	x_{j+1}	x_{j+2}	x_{j+3}	x_{j+4}	Proposed algorithm	GCK algorithm
$y(4,0,j)$	1	1	1	1		$y(4,0,j+1)$	$y(4,0,j+1)$
$y(4,0,j+1)$		1	1	1	1	$= y(4,0,j) - x_j + x_{j+4}$	$= y(4,0,j) - x_j + x_{j+4}$
$y(4,2,j)$	1	-1	-1	1		$y(4,1,j+1)$	$y(4,1,j+2)$
$y(4,1,j+1)$		1	1	-1	-1	$= -y(4,2,j) + x_j - x_{j+4}$	$= y(4,0,j) - y(4,0,j+2) - y(4,1,j)$
$y(4,1,j)$	1	1	-1	-1		$y(4,2,j+1)$	$y(4,2,j+2)$
$y(4,2,j+1)$		1	-1	-1	1	$= y(4,1,j) - x_j + x_{j+4}$	$= y(4,1,j+1) - y(4,1,j+2) - y(4,2,j+1)$
$y(4,3,j)$	1	-1	1	-1		$y(4,3,j+1)$	$y(4,3,j+2)$
$y(4,3,j+1)$		1	-1	1	-1	$= -y(4,3,j) + x_j - x_{j+4}$	$= y(4,3,j) - y(4,2,j) - y(4,2,j+2)$

Table 2.1: *Fast Algorithm when Window Size is 4.*

computed from those in window j and $d_N(j)$. Therefore, we have

$$\begin{bmatrix} y(4,0,j+1) \\ y(4,1,j+1) \\ y(4,2,j+1) \\ y(4,3,j+1) \end{bmatrix} = \begin{bmatrix} y(4,0,j) \\ -y(4,2,j) \\ y(4,1,j) \\ -y(4,3,j) \end{bmatrix} + \begin{bmatrix} -d_4(j) \\ d_4(j) \\ -d_4(j) \\ d_4(j) \end{bmatrix}, \tag{2.9}$$

and Fig. 2.2 shows the signal flow diagram.

Thus, after obtaining $\vec{\mathbf{y}}^{(4,j)}$, the proposed algorithm obtains $\vec{\mathbf{y}}^{(4,j+1)}$ by five additions as shown in (2.8) and (2.9), whereas the algorithm in [32] requires six additions and the GCK algorithm requires eight additions.

2.3.2 Fast Algorithm for Window Size 8

The proposed algorithm and the GCK algorithm [32] for window size of 8 are described in Table 2.2. The proposed algorithm computes $\vec{\mathbf{y}}^{(8,j+2)}$ for $j = 0, 1, \ldots J-8$, which is the WHT projection value vector in window $j+2$, using $\vec{\mathbf{y}}^{(8,j)}$, which is the computed projection value vector in window j.

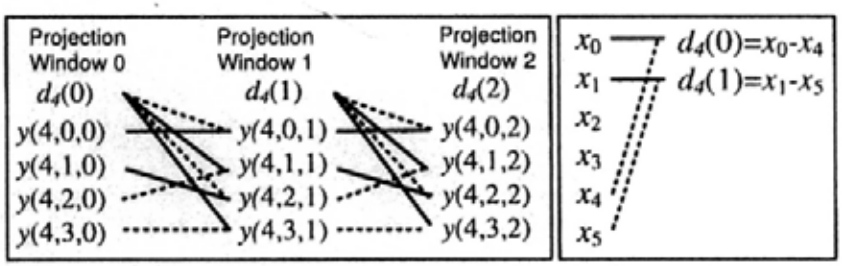


Figure 2.2: *Signal flow diagram of the Bottom up algorithm for window size equal 4.*

	x_j	x_{j+1}	x_{j+2}	x_{j+3}	x_{j+4}	x_{j+5}	x_{j+6}	x_{j+7}	x_{j+8}	x_{j+9}	Proposed algorithm	GCK
$y(8,0,j)$	1	1	1	1	1	1	1	1			$y(8,0,j+2)$	$y(8,0,j+2)$
$y(8,0,j+2)$			1	1	1	1	1	1	1	1	$=y(8,0,j)-x_j-x_{j+1}+x_{j+8}+x_{j+9}$	$=y(8,0,j+1)-x_{j+1}+x_{j+9}$
$y(8,2,j)$	1	1	-1	-1	-1	-1	1	1			$y(8,1,j+2)$	$y(8,1,j+2)$
$y(8,1,j+2)$			1	1	1	1	-1	-1	-1	-1	$=-y(8,2,j)+x_j+x_{j+1}-x_{j+8}-x_{j+9}$	$=y(8,0,j-2)-y(8,0,j+2)-y(8,1,j-2)$
$y(8,1,j)$	1	1	1	1	-1	-1	-1	-1			$y(8,2,j+2)$	$y(8,2,j+2)$
$y(8,2,j+2)$			1	1	-1	-1	-1	-1	1	1	$=y(8,1,j)-x_j-x_{j+1}+x_{j+8}+x_{j+9}$	$=y(8,1,j)-y(8,1,j+2)-y(8,2,j)$
$y(8,3,j)$	1	1	-1	-1	1	1	-1	-1			$y(8,3,j+2)$	$y(8,3,j+2)$
$y(8,3,j+2)$			1	1	-1	-1	1	1	-1	-1	$=-y(8,3,j)+x_j+x_{j+1}-x_{j+8}-x_{j+9}$	$=y(8,3,j-2)-y(8,2,j+2)-y(8,2,j-2)$
$y(8,4,j)$	1	-1	-1	1	1	-1	-1	1			$y(8,4,j+2)$	$y(8,4,j+2)$
$y(8,4,j+2)$			1	-1	-1	1	1	-1	-1	1	$=-y(8,4,j)+x_j-x_{j+1}-x_{j+8}+x_{j+9}$	$=y(8,3,j+1)-y(8,3,j+2)-y(8,4,j+1)$
$y(8,6,j)$	1	-1	1	-1	-1	1	-1	1			$y(8,5,j+2)$	$y(8,5,j+2)$
$y(8,5,j+2)$			1	-1	-1	1	-1	1	1	-1	$=y(8,6,j)-x_j+x_{j+1}+x_{j+8}-x_{j+9}$	$=y(8,4,j-2)-y(8,4,j+2)-y(8,5,j-2)$
$y(8,5,j)$	1	-1	-1	1	-1	1	1	-1			$y(8,6,j+2)$	$y(8,6,j+2)$
$y(8,6,j+2)$			1	-1	1	-1	-1	1	-1	1	$=-y(8,5,j)+x_j-x_{j+1}-x_{j+8}+x_{j+9}$	$=y(8,6,j)-y(8,5,j)-y(8,5,j+2)$
$y(8,7,j)$	1	-1	1	-1	1	-1	1	-1			$y(8,7,j+2)$	$y(8,7,j+2)$
$y(8,7,j+2)$			1	-1	1	-1	1	-1	1	-1	$=y(8,7,j)-x_j+x_{j+1}+x_{j+8}-x_{j+9}$	$=y(8,7,j-2)-y(8,6,j-2)-y(8,6,j+2)$

Table 2.2: *Fast Algorithm when Window Size is 8.*

Define $t_{N/4}(i,j)$ for $i = 0,\ldots,N/4-1, j = 0,1,\ldots,J-5N/4$ as:

$$t_{N/4}(i,j) = y(N/4,i,j) - y(N/4,i,j+N). \tag{2.10}$$

For $N = 8$, we have

$$\begin{bmatrix} t_2(0,j) \\ t_2(1,j) \end{bmatrix} = \begin{bmatrix} y(2,0,j) \\ y(2,1,j) \end{bmatrix} - \begin{bmatrix} y(2,0,j+8) \\ y(2,1,j+8) \end{bmatrix}. \tag{2.11}$$

From (2.4) and then (2.8), we have

$$\begin{aligned} \begin{bmatrix} t_2(0,j) \\ t_2(1,j) \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_j \\ x_{j+1} \end{bmatrix} - \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_{j+8} \\ x_{j+9} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_j - x_{j+8} \\ x_{j+9} - x_{j+9} \end{bmatrix} = \mathbf{M}^{(2)} \begin{bmatrix} d_8(j) \\ d_8(j+1) \end{bmatrix} \end{aligned} \tag{2.12}$$

As given by (2.12), $t_2(i,j)$ is the ith order-2 WHT projection value of $[d_8(j)\ d_8(j+1)]^T$. According to Table 2.2 and (2.12), WHT projection value vector in window $j+2$ can

Step a $d_8(j+1) = x_{j+1} - x_{j+9}$.
- One addition is required.

Step b $t_2(0,j) = [1, 1]\,[\,d_8(j), d_8(j+1)]^t$,
$t_2(1,j) = [1, -1]\,[d_8(j), d_8(j+1)]^t$.
- Two additions are required. Note that $d_8(j)$ was obtained during computation of $\vec{Y}_8(j+1)$ in *Step a*.

Step c

$$y(8,i,j+2) = \begin{cases} (-1)^{v+i}\{y(8,i,j) - t_2(v,j)\}, i = 0,3,4,7 \\ (-1)^{v+i}\{y[8, i-(-1)^i, j] - t_2(v,j)\}, i = 1,2,5,6 \end{cases}$$

where $v = \lfloor i/4 \rfloor$.
- Eight additions are required in this step for $i = 0, 1, \ldots, 7$.

Table 2.3: *Computation of All Order-8 Projection Values in Window $j+2$.*

be computed from those in window j as well as $t_2(i,j)$ as follow:

$$\begin{bmatrix} y(8,0,j+2) \\ y(8,1,j+2) \\ y(8,2,j+2) \\ y(8,3,j+2) \\ y(8,4,j+2) \\ y(8,5,j+2) \\ y(8,6,j+2) \\ y(8,7,j+2) \end{bmatrix} = \begin{bmatrix} y(8,0,j) \\ -y(8,2,j) \\ y(8,1,j) \\ -y(8,3,j) \\ -y(8,4,j) \\ y(8,6,j) \\ -y(8,5,j) \\ y(8,7,j) \end{bmatrix} + \begin{bmatrix} -t_2(0,j) \\ t_2(0,j) \\ -t_2(0,j) \\ t_2(0,j) \\ t_2(1,j) \\ -t_2(1,j) \\ t_2(1,j) \\ -t_2(1,j) \end{bmatrix}. \tag{2.13}$$

In summary, (2.13) can be represented by

$$y(N,i,j+2) = \begin{cases} (-1)^{v+i}[y(8,i,j) + t_2(v,j)], & i = 0,3,4,7 \\ (-1)^{v+i}[y(8,i-(-1)^i,j) + t_2(v,j)], & i = 1,2,5,6, \end{cases} \tag{2.14}$$

where $v = \lfloor i/4 \rfloor$.

Table 2.3 gives the three steps for computing $\vec{y}^{(8,j+2)}$ from $\vec{y}^{(8,j)}$ as well as $\vec{y}^{(8,j+1)}$ and the corresponding number of operations required.

Therefore, the proposed algorithm requires 11 additions, whereas the algorithm in [32] requires 14 additions for obtaining the eight projection values in $\vec{y}^{(8,j+2)}$. The GCK algorithm requires 16 additions.

2.4 Fast Algorithm for WHT on Sliding Windows for Window Size N

2.4.1 The Algorithm

Let $\mathbf{D}^{(N)}$ be the order-N reverse-identity matrix, i.e., elements at the reverse-diagonal positions are 1 and 0 at others. For example, $\mathbf{D}^{(4)}$ is

$$\mathbf{D}^{(4)} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}. \tag{2.15}$$

The equation below is proved in the Appendix:

$$\begin{bmatrix} y(N,8i+0,j) \\ y(N,8i+1,j) \\ y(N,8i+2,j) \\ y(N,8i+3,j) \\ y(N,8i+4,j) \\ y(N,8i+5,j) \\ y(N,8i+6,j) \\ y(N,8i+7,j) \end{bmatrix} = \begin{bmatrix} \mathbf{M}^{(4)} & 0_{4\times4} \\ 0_{4\times4} & \mathbf{D}^{(4)}\mathbf{M}^{(4)} \end{bmatrix} \begin{bmatrix} y(N/4,2i,j) \\ y(N/4,2i,j+N/4) \\ y(N/4,2i,j+2N/4) \\ y(N/4,2i,j+3N/4) \\ y(N/4,2i+1,j) \\ y(N/4,2i+1,j+N/4) \\ y(N/4,2i+1,j+2N/4) \\ y(N/4,2i+1,j+3N/4) \end{bmatrix}, \tag{2.16}$$

for $i = 0, 1, \ldots N/8 - 1$.

Hence, order-N WHT is partitioned into $N/4$ groups of order-4 WHT. Utilizing the method in (2.9) for each of the order-4 WHT in (2.16), WHT projection values in window $j + N/4$ can be computed using projection values in window j as well as

$t_{N/4}(2i,j)$ and $t_{N/4}(2i+1,j)$ as follows:

$$\begin{bmatrix} y(N,8i+0,j+N/4) \\ y(N,8i+1,j+N/4) \\ y(N,8i+2,j+N/4) \\ y(N,8i+3,j+N/4) \\ y(N,8i+4,j+N/4) \\ y(N,8i+5,j+N/4) \\ y(N,8i+6,j+N/4) \\ y(N,8i+7,j+N/4) \end{bmatrix} = \begin{bmatrix} y(N,8i+0,j) \\ -y(N,8i+2,j) \\ y(N,8i+1,j) \\ -y(N,8i+3,j) \\ -y(N,8i+4,j) \\ y(N,8i+6,j) \\ -y(N,8i+5,j) \\ y(N,8i+7,j) \end{bmatrix} + \begin{bmatrix} -t_{N/4}(2i+0,j) \\ t_{N/4}(2i+0,j) \\ -t_{N/4}(2i+0,j) \\ t_{N/4}(2i+0,j) \\ t_{N/4}(2i+1,j) \\ -t_{N/4}(2i+1,j) \\ t_{N/4}(2i+1,j) \\ -t_{N/4}(2i+1,j) \end{bmatrix}, \quad (2.17)$$

where $t_{N/4}(i,j)$ is defined in (2.10). Equation (2.17), which is for order-N WHT, becomes (2.13) when $N=8$.

Let us first consider the computation of $t_{N/4}(i,j)$ in (2.17). Utilizing the $d_N(j)$ in (2.8), we define

$$\vec{\mathbf{d}}^{(N/4,j)} = [d_N(j), d_N(j+1), \ldots, d_N(j+N/4-1)]^T = \vec{\mathbf{x}}^{(N/4,j)} - \vec{\mathbf{x}}^{(N/4,j+N)}. \quad (2.18)$$

From (2.4), (2.10), and then (2.18), we have

$$\begin{aligned} t_{N/4}(i,j) &= y(N/4,i,j) - y(N/4,i,j+N) \\ &= \vec{\mathbf{m}}^{(N/4,i)^T} \vec{\mathbf{x}}^{(N/4,j)} - \vec{\mathbf{m}}^{(N/4,i)^T} \vec{\mathbf{x}}^{(N/4,j+N)} \\ &= \vec{\mathbf{m}}^{(N/4,i)^T} [\vec{\mathbf{x}}^{(N/4,j)} - \vec{\mathbf{x}}^{(N/4,j+N)}] \\ &= \vec{\mathbf{m}}^{(N/4,i)^T} \vec{\mathbf{d}}^{(N/4,j)}. \end{aligned} \quad (2.19)$$

Equation (2.19) shows that $t_{N/4}(i,j)$ is the ith projection value of the order-$N/4$ WHT of $\vec{\mathbf{d}}^{(N/4,j)}$. When $N=8$, (2.19) becomes (2.12).

The signal flow diagram in Fig. 2.3 depicts the computation of order-N WHT using (2.17)-(2.19). Table 4 describes the computation of $\vec{\mathbf{y}}^{(N,j+N/4)}$ from $\vec{\mathbf{y}}^{(N,j)}$ and the corresponding number of operations as well as memory required. Since at most size $2J$ memory is required for the proposed algorithm at each step, the memory required for the proposed algorithm is $2J$, which is the same as the GCK algorithm.

This chapter focuses on 1D WHT. However, it is easy to extend the proposed 1D WHT algorithm to higher dimensions. For example, when the 2D WHT of size $N \times N$

Overall procedure:
For each j {*Step a*;}
For each i: {
  For each j { *Step b*; }
  For each j { *Step c*; }
}

Step a: Compute $d_N(j) = x_j - x_{j+N}$. This step provides the $\vec{\mathbf{d}}^{(N/4,j)}$ in (2.18) for the computation of $t_{N/4}(i,j)$ in *Step b*.
Analysis: One addition per window is required. Size $2J$ memory is required for storing $d_N(j)$ and the input data x_j for $j = 0, \ldots, J-1$. Note that x_j will not be used in the following steps.
Step b: Compute $t_{N/4}(\lfloor i/4 \rfloor, j) = \vec{\mathbf{m}}^{(N/4,\lfloor i/4 \rfloor)^T} \vec{\mathbf{d}}^{(N/4,j)}$. This step provides the $t_{N/4}$ in (2.17) for the computation of $y(N,i,j+N/4)$ in *Step c*. We can use the GCK algorithm in [32] for computation.
Analysis: $N/2$ additions per window are required for the $N/4$ values of $t_{N/4}(\lfloor i/4 \rfloor, j)$ from $\vec{\mathbf{d}}^{(N/4,j)}$ for given j. As stated in [32], size $2J$ memory is required by GCK.
Step c: Obtain $y(N,i,j+N/4)$ using (2.17). Note that the $y(N,i,j)$ in (2.17) is computed previously.
Analysis: N additions per window are required for the N values of i. Size J memory is required for storing the $t_{N/4}(i,j)$ for $j = 0, \ldots, J-1$ which are computed in *Step b*; size $O(N)$ memory is required for storing projection values $y(N/4,i,a)$ for $j \le a < j+N/4$ required in the right hand side of (2.17) for the given i. Since we have $N \ll J$ for most cases, the memory requirement is less than $2J$ in this step.

Table 2.4: *Computation of Order-N WHT.*

is computed, our algorithm in Table 2.4 can first use GCK for computing the WHT of size $N \times N/4$ in *Step b*, and then, use *Step c* to obtain the projection values of size $N \times N$. In this way, we require 1 addition in *Step a*, $N^2/2$ additions in *Step b*, and N^2 additions in *Step c*, i.e., $1.5N^2$ additions in all. In this way, the proposed algorithm

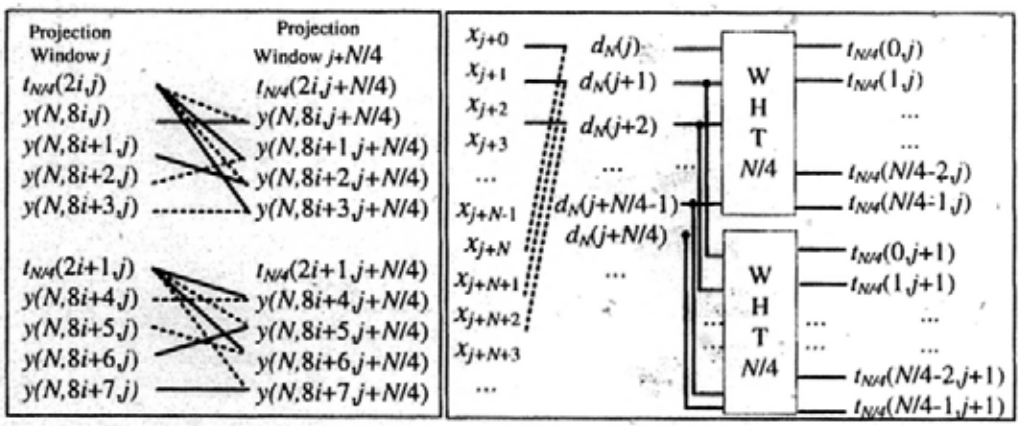


Figure 2.3: *Signal flow diagram of the bottom up algorithm for order-N sequency WHT.*

Size	4	8	16	32	N
GCK	8	16	32	64	$2N$
Proposed	5	11	25	49	$3N/2+1$

Table 2.5: *Numbers of Additions Required by the GCK Algorithm and the Proposed Algorithm for All Projection Values of Order-N WHT.*

requires 1.5 additions per window per projection value independent of dimension. In comparison, the GCK algorithm requires two additions per window per projection value independent of dimension. In Section 2.6, we will show the experimental result that uses the fast algorithm for 2D WHT.

2.5 Computational Requirement of the Proposed Fast Algorithms for Window Size N

2.5.1 When All Projection Values Are Computed

Let the total number of additions for obtaining $\vec{\mathbf{y}}^{(N,j+4)}$ be $B(N,N)$. According to the analysis in Table 2.4, we require one addition in *Step a*, $N/2$ additions in *Step b*, and N additions in *Step c*. So, we have

$$B(N,N) = 1 + N/2 + N = 3N/2 + 1. \tag{2.20}$$

The number of additions required for the GCK algorithm and the proposed algorithm is summarized in Table 2.5, which shows that the proposed algorithm requires about $3N/2$ additions, while the GCK algorithm requires $2N$ additions. The number of additions required by our algorithm for order-4 and order-8 WHT are 5 and 11, respectively, because we can use direct computation instead of the GCK for calculating $t_{N/4}(i,j)$ in the *Step b* of Table 2.4. For example, if $N = 4$, then $t_{N/4}(i,j) = d_4(j)$ and no computation is required in *Step b* of Table 2.4 for obtaining $t_{N/4}(i,j)$.

2.5.2 When Not All Projection Values Are Computed

In many applications, not all projection values are required. In this part, we analyze the computational requirement when only the first P projection values are computed for window size N. Specifically, we shall derive the number of additions for the computation of $y(N,0,j), y(N,1,j), \ldots, y(N,P-1,j)$ for $j = N/4, N/4+1, \ldots, J-N$. Here, we shall not consider the case when $j < N/4$ because the computational complexity is

The overall procedure and the three steps are the same as that in Table 2.4. The only difference is that the total number of i is P now. **Analysis:** ***Step a***: 1 addition per window is required in this step. ***Step b***: $2 \cdot \lceil P/4 \rceil$ additions are required in this step using the GCK for the $\lceil P/4 \rceil$ values of $t_{N/4}(\lfloor i/4 \rfloor, j)$. ***Step c***: If $P\%4 \equiv 2$ (for example P is 2 or 6), for the computation in (2.17), the proposed algorithm needs to compute $y(N, 4 \cdot \lfloor P/4 \rfloor + 2, j)$ for $y(N, 4 \cdot \lfloor P/4 \rfloor + 1, j + N/4)$; So $P+1$ additions are required if the $P\%4 \equiv 2$; Otherwise, P additions are required.

Table 2.6: *Computation of order-N WHT When Not All Projection Values are Required.*

negligible as $N \ll J$ in most cases. A zero-padding approach dealing with the cases when $j < N/4$ is introduced in [2].

Let the number of additions per window required to obtain $y(N, i, j + N/4)$ for $i = 0, \ldots, P-1$; $j = 0, 1, \ldots, J - 5N/4$ be $B_N(P)$. Table 2.6 lists the steps and the corresponding number of additions required. As shown in Table 2.6, we require one addition in *Step a*, $2 \cdot \lceil P/4 \rceil$ additions in *Step b*, and at most $P+1$ additions in *Step c*. The number of additions required for obtaining P projection values in order-N WHT using the proposed algorithm as given in Table 2.6 has the following inequality:

$$B(N, P) \leq 1 + 2 \cdot \lceil P/4 \rceil + P + 1 \leq \lceil 3P/2 \rceil + 3. \tag{2.21}$$

The computation required is about 1.5 additions/pixel/kernel using the proposed algorithm.

2.6 Experimental Results

To investigate the computational efficiency of the proposed algorithm for pattern matching in practical applications, block matching in motion estimation is utilized. Block matching in motion estimation using fast WHT was carried out on the first 200 frames of a video sequence "tempete" which has a resolution of 352×288. The experiment considers the execution time required for obtaining different numbers of WHT projection values, which ranges from 1 to 20. The proposed algorithm is compared with the algorithm in [2], which utilized the GCK algorithm.

In a similar experiment reported in [2], two projection value computation orders

were used. They are the "snake order" and "increasing frequency order." Fig. 2.4 shows the ordering of the first 20 projection values of these two orders. The percentage of the time required by the proposed algorithm with respect to the GCK algorithm is given in Fig. 2.5. The proposed algorithm outperforms the GCK algorithm when the number of projections is greater than 5. As the proposed algorithm computes three or four projection values together to save computation, whereas the GCK algorithm does not, so the percentage of computational time saved by the proposed algorithm in comparison with the GCK algorithm depends on the number of projections. Generally, the proposed algorithm achieves a higher saving when most projection values to be computed can take advantage of this property. This is why, when the number of projection values approaches 13 and 16 for snake order, the proposed algorithm requires the least percentage of time compared with the GCK algorithm. When the number of projection values is less than five, the proposed algorithm requires more computational time because projection values cannot be grouped together for computation. Therefore, we would suggest the use of the GCK algorithm when the number of projection values is less than five.

2.7 Summary

This chapter proposes a fast computational algorithm for Walsh Hadamard Transform on sliding windows, which requires about 1.5 additions per projection value per window. The computational time of the proposed algorithm is about 75 percent that of the GCK algorithm. In cases where not all projection values are needed, the proposed algorithm can outperform the GCK algorithm when the number of projection values is five or above. The proposed algorithm achieves its high efficiency in the computation of order-N WHT by using order-4 and order-$N/4$ WHT. This chapter provides fast

0	1	8	9	
3	2	7	10	
4	5	6	11	
15	14	13	12	
16	17	18	19	

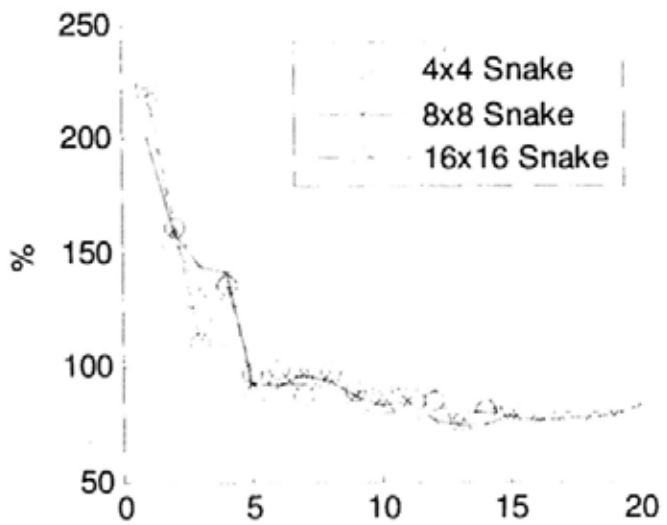
(a) Snake order

0	2	5	10	16
1	3	7	12	18
4	6	9	14	
8	11	13	19	
15	17			

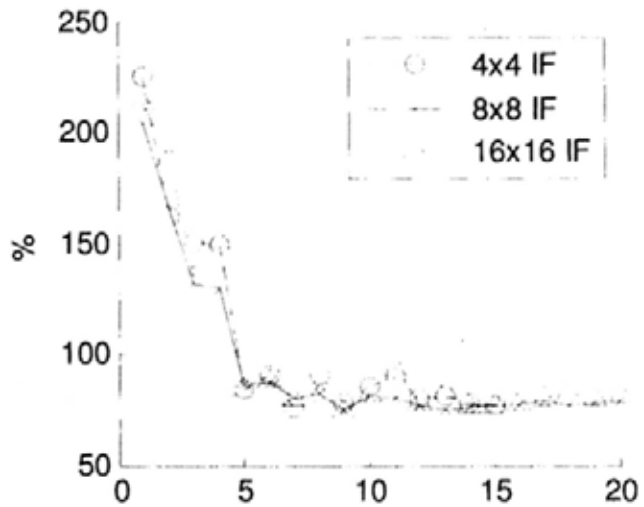
(b) Increasing frequency order

Figure 2.4: *Two different projection orders.*

algorithm for 1D WHT. In the future, we are going to seek an even faster algorithm. We will also try to see if there exists a superset of GCK that can be computed by a constant number of additions per window per projection value independent of the size and dimension of the transform.



(a) Snake order



(b) Increasing frequency order

Figure 2.5: *The percentage of time required by our algorithm with respect to GCK algorithm when different number of projection values are computed, where* ***Snake*** *stands for the snake order and* ***IF*** *stands for the increasing frequency order. The experiment is implemented on a 2.13GHz PC using C on windows XP system with compiling environment VC 6.0.*

2.8 Appendix A: Proof for (2.16)

This appendix provides the proof for (2.16). Except for this appendix, sequency order is used for representing WHT. In this appendix, dyadic-ordered WHT will be utilized for proving (2.16). Natural order-N WHT can be represented by:

$$\mathbf{M}^{(N)} = \mathbf{M}^{(2)} \otimes \mathbf{M}^{(N/2)}, \tag{2.22}$$

where $\otimes$ is the Kronecker product ($A \otimes B$ is a $mp \times nq$ matrix composed of the $m \times n$ blocks $(a_{i,j}B)$) and

$$\mathbf{M}^{(2)} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \tag{2.23}$$

Both sequency and dyadic orders [52] are the reordering form of the natural order for WHT. Here we denote $\mathbf{M}_Z^{(N)}$ as the order-N sequency-ordered WHT matrix; denote $\mathbf{M}_D^{(N)}$ as the order-N dyadic-ordered WHT matrix and:

$$\mathbf{M}_D^{(N)} = \left[\vec{\mathbf{m}}_D^{(N,0)} \; \vec{\mathbf{m}}_D^{(N,1)} \; , \ldots , \; \vec{\mathbf{m}}_D^{(N,N-1)} \right]^T \tag{2.24}$$

where $\vec{\mathbf{m}}_D^{(N,i)}$ is the ith WHT basis vector. The binary vector representation of i in (2.24) is $[i_1, i_2, \ldots, i_{G_M}]^T$, where i_k are 0 or 1 for $k = 1, \ldots, G_M$ and:

$$i = 2^{G_M-1} i_1 + 2^{G_M-2} i_2 + \ldots + 2 i_{G_M-1} + i_{G_M}. \tag{2.25}$$

For dyadic-ordered WHT, for $b = 0, \ldots, a-1, a = 2, 4, 8, \ldots, N$, we have:

$$\vec{\mathbf{m}}_D^{(N,ai+b)} = \vec{\mathbf{m}}_D^{(a,b)} \otimes \vec{\mathbf{m}}_D^{(N/a,i)} \tag{2.26}$$

Let i^Z and i^D be indices of sequency-ordered and dyadic-ordered WHT respectively. As pointed out in [52], the relationship between the binary vector representation of i^Z

and i^D is:

$$i^Z = [W_{D,Z}]_{G_M} i^D,$$

$$\text{where } [W_{D,Z}]_{G_M} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 1 & 1 & \dots & 1 \end{bmatrix} \tag{2.27}$$

According to (2.27), if $\vec{\mathbf{m}}_Z^{(N,ai^Z+b^Z)} = \vec{\mathbf{m}}_D^{(N,ai^D+b^D)}$, where b^Z, $b^D < a$, $a = 2^k$, then i^Z is only decided by i^D. So we have:

$$\vec{\mathbf{m}}_Z^{(N,ai^Z+b_i^Z)} = \vec{\mathbf{m}}_D^{(N,ai^D+b^D)} \Rightarrow \vec{\mathbf{m}}_Z^{(N,i^Z)} = \vec{\mathbf{m}}_D^{(N,i^D)}. \tag{2.28}$$

Denote $f(b,a,i)$ as:

$$f(b,a,i) = \begin{cases} b, & i \text{ is an even number}, \\ a-1-b, k & i \text{ is an odd number}, \end{cases} \tag{2.29}$$

The b_i^Z in (2.28) is decided by both i^Z and b^D:

$$b_i^Z = f([W_{D,Z}]b^D, a, i^Z), \tag{2.30}$$

where the size of $[W_{D,Z}]$ is $\log_2 a \times \log_2 a$.

It is obvious that $f[f(b,a,i),a,i] = b$, so we have:

$$[W_{D,Z}]b^D = f[f([W_{D,Z}]b^D, a, i^Z), a, i^Z] = f(b_i^Z, a, i^Z). \tag{2.31}$$

$\vec{\mathbf{m}}_Z^{(N,ai^Z+b_i^Z)}$ can be represented as follows using (2.26), (2.28) and (2.31):

$$\begin{aligned} \vec{\mathbf{m}}_Z^{(N,ai^Z+b_i^Z)} &= \vec{\mathbf{m}}_D^{(N,ai^D+b^D)} = \vec{\mathbf{m}}_D^{(a,b^D)} \otimes \vec{\mathbf{m}}_D^{(N/a,i^D)} \\ &= \vec{\mathbf{m}}_Z^{(a,[W_{D,Z}]b^D)} \otimes \vec{\mathbf{m}}_Z^{(N/a,i^Z)} = \vec{\mathbf{m}}_Z^{(a,f(b_i^Z,a,i^Z))} \otimes \vec{\mathbf{m}}_Z^{(N/a,i^Z)}. \end{aligned} \tag{2.32}$$

According to (2.32), we have:

$$\vec{\mathbf{m}}_Z^{(N,ai+b)} = \vec{\mathbf{m}}_Z^{(a,f(b,a,i))} \otimes \vec{\mathbf{m}}_Z^{(N/a,i)}. \tag{2.33}$$

Therefore, $y_Z(N, ai+b, j)$ can be represented as follows:

$$
\begin{aligned}
y_Z(N, ai+b, j) &= \{\vec{\mathbf{m}}_Z^{(a,f(b,a,i))} \otimes \vec{\mathbf{m}}_Z^{(N/a,i)}\}\vec{\mathbf{x}}^{(N,j)} \\
&= \left\{[\vec{\mathbf{m}}_Z^{(a,f(b,a,i))}\mathbf{I}_a] \otimes [\mathbf{I}_1 \vec{\mathbf{m}}_Z^{(N/a,i)}\right\} \vec{\mathbf{x}}^{(N,j)} \\
&= \left\{[\mathbf{I}_1 \otimes \vec{\mathbf{m}}_Z^{(a,f(b,a,i))}][\mathbf{I}_a \otimes \vec{\mathbf{m}}_Z^{(N/a,i)}]\right\} \vec{\mathbf{x}}^{(N,j)} \\
&= \vec{\mathbf{m}}_Z^{(a,f(b,a,i))} \left\{ \begin{bmatrix} \vec{\mathbf{m}}_Z^{(N/a,i)} & & & \\ & \vec{\mathbf{m}}_Z^{(N/a,i)} & & \\ & & \cdots & \\ & & & \vec{\mathbf{m}}_Z^{(N/a,i)} \end{bmatrix} \vec{\mathbf{x}}^{(N,j)} \right\} \\
&= \vec{\mathbf{m}}_Z^{(a,f(b,a,i))} \begin{bmatrix} y_Z(N/a, i, j) \\ y_Z(N/a, i, j+N/a) \\ \cdots \\ y_Z(N/a, i, j+N-N/a) \end{bmatrix}_{a\times 1}
\end{aligned}
\tag{2.34}
$$

The following equation is valid using (2.34):

$$\begin{bmatrix} y_Z(N,2ai^Z+0,j) \\ y_Z(N,2ai^Z+1,j) \\ \cdots \\ y_Z(N,2ai^Z+a-1,j) \\ y_Z(N,2ai^Z+a,j) \\ y_Z(N,2ai^Z+a+1,j) \\ \cdots \\ y_Z(N,2ai^Z+2a-1,j) \end{bmatrix} = \begin{bmatrix} \mathbf{M}_Z^{(a)} \begin{bmatrix} y_Z(N/a,2i^Z,j) \\ y_Z(N/a,2i^Z,j+N/a) \\ \cdots \\ y_Z(N/a,2i^Z,j+N-N/a) \end{bmatrix} \\ \mathbf{D}^{(a)}\mathbf{M}_Z^{(a)} \begin{bmatrix} y_Z(N/a,2i^Z+1,j) \\ y_Z(N/a,2i^Z+1,j+N/a) \\ \cdots \\ y_Z(N/a,2i^Z+1,j+N-N/a) \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{M}_Z^{(a)} & \\ & \mathbf{D}^{(a)}\mathbf{M}_Z^{(a)} \end{bmatrix} \begin{bmatrix} y_Z(N/a,2i^Z,j) \\ y_Z(N/a,2i^Z,j+N/a) \\ \cdots \\ y_Z(N/a,2i^Z,j+N-N/a) \\ y_Z(N/a,2i^Z+1,j) \\ y_Z(N/a,2i^Z+1,j+N/a) \\ \cdots \\ y_Z(N/a,2i^Z+1,j+N-N/a) \end{bmatrix} \tag{2.35}$$

Equ. (2.16) is valid when $a = 4$ in (2.35).

Chapter 3

Performance Evaluation of Full Search Equivalent Pattern Matching Algorithms

3.1 Introduction

In the previous chapter, we propose a fast WHT algorithm. The fast WHT algorithm can be applied for FS equivalent pattern matching. Recently, there are many FS equivalent algorithms proposed [1; 30–33; 44; 45]. Motivated by the intense research activity recently developed in FS equivalent pattern matching, this chapter aims to compare and analyze state-of-the-art FS-equivalent algorithms for pattern matching in different conditions. For this aim, this chapter selects the following five algorithms which are recent and have been shown to yield notable speed-ups against the FS approach.

1. Alkhansari's Low Resolution Pruning (LRP) algorithm [30];
2. Tombari *et al.*'s Increasing Dissimilarity Approximation (IDA) algorithm [31];
3. Hel-Or and Hel-Or's projection based algorithm (PWHT) using Walsh-Hadamard Transform (WHT) [1];
4. Ben-Artzi and Hel-Ors' projection based algorithm using GCK (PGCK) [32];
5. Ouyang and Cham's projection based algorithm using fast WHT (FWHT) [33].

Table 3.1 summarizes the compared algorithms together with the corresponding abbreviations. In addition to the FS, the Fast Fourier Transform (FFT) approach is also used as a benchmark of comparison (in particular, we use the OpenCV implementation [53]).

For the purpose of comparing algorithms, we consider both execution time and computational complexity of the algorithms. In particular, the execution time is evaluated

LRP	IDA	PWHT	PGCK	FWHT	FFT
[30]	[31]	[1]	[32]	[33]	[53]

Table 3.1: *Abbreviations and references of compared algorithms*

over different platforms since it has some dependencies on the hardware characteristics, while the computation complexity is more general. Moreover, though execution time has been used for showing the computational efficiency of IDA, PWHT, PGCK and FWHT in [1; 31–33], computational analysis of these algorithms is not available yet. Hence, this chapter analyzes the computational complexity of IDA, PWHT, PGCK, FWHT.

Our intention is that the datasets and tests used in our evaluation will be a benchmark for testing future pattern matching algorithms, and that this analysis concerning the performance of state-of-the-art algorithms could inspire new fast algorithms. The datasets and the code to carry out the experiments are available online [1].

This chapter is organized as follows. Section 3.2 presents a unified framework that can represent all evaluated algorithms for further analysis. Based on this unified framework, the computational complexity of evaluated algorithms is analyzed in Section 3.3. Then, the algorithms are compared quantitatively using the datasets described in Section 3.4, which account for different sizes of images and patterns as well as for distortions caused by different types of noise. The testing environment and evaluation criterion will also be described in Section 3.4. Section 3.5 illustrates the experimental results. Finally, Section 3.6 presents a discussion and draws conclusions.

3.2 A Unified Framework for Pattern Matching Algorithms

In this section, we first introduce a unified framework for fast FS-equivalent pattern matching. The links and differences among the algorithms compared in this chapter are then analyzed within this framework.

As highlighted in Table 3.2, the proposed framework consists of two steps:

- *Step a*: Mismatching candidate windows are eliminated from set_{can}. This step is called rejection step.
- *Step b*: The remaining candidate windows in set_{can} undergo FS for finding out

[1]The website will be available after acceptance of the paper

Overall procedure:
Initially, set_{can} contains all candidate windows $\vec{\mathbf{x}}_w^{(N,j)}$;
Step a (Rejection Step):
For k:=1 to N_{Maxk}:
{*Step a.1*; *Step a.2*; *Step a.3*.}
Step b (FS-Step):
For each candidate window $\vec{\mathbf{x}}_w^{(N,j)}$ in set_{can}:
{ Use $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_p^p$ to find out if $\vec{\mathbf{x}}_w^{(N,j)}$ matches $\vec{\mathbf{x}}_t^{(N)}$.}

Step a.1:
For $\vec{\mathbf{x}}_w^{(N,j)}$ in set_{can}: {Obtain $f_{low}(k,j)$.}
Step a.2:
For $\vec{\mathbf{x}}_w^{(N,j)}$ in set_{can}: {If $f_{low}(k,j) > T$, then remove $\vec{\mathbf{x}}_w^{(N,j)}$ from set_{can}.}
Step a.3:
$N_k = k$; if $Cond_{Ter}$ is true, then goto *Step b*.

Table 3.2: *Unified framework for pattern matching using lower bound.*

the matching windows. This step is called FS-step.

In this framework, both FS and FFT directly evaluate the distance $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_p^p$ for all candidate windows to find the matching windows. This corresponds to skipping the rejection step and starting from FS-step in Table 3.2. The FFT approach is only different from FS in the computation of $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_p^p$.

On the other hand, algorithms IDA, LRP, PWHT, PGCK and FWHT start from the rejection step. The rejection step is an iteration of k, where k increases from 1, and comprises three sub-steps. In *Step a.1*, a lower bounding function $f_{low}(k,j)$ is evaluated such that $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_p^p \geq f_{low}(k,j)$. In *Step a.2*, if $f_{low}(k,j) \geq T$, then $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_p^p \geq T$ and we can safely prune $\vec{\mathbf{x}}_w^{(N,j)}$ from set_{can}, $f_{low}(k,j) \geq T$ being the rejection condition. In *Step a.3*, the loop of k terminates when a given termination condition, denoted as $Cond_{Ter}$, is satisfied. Throughout the rejection step, each candidate window undergoes checking of a succession of rejection conditions $f_{low}(k,j) \geq T$ in each loop of k, until either it is pruned or the rejection step finishes. There are two conditions under which *Step a* terminates: 1) k reaches a sufficient number which is denoted as N_{Maxk}; 2) the percentage of remaining candidate windows in iteration $k+1$, denoted as $Per_{can}^{(k+1)}$, is below certain threshold denoted as ϵ. The second termination condition, corresponding to $Cond_{Ter}$ in *Step a.3*, is a strategy used by Hel-Or and Hel-Or in [1] because it turns out faster to directly calculate the actual distance than

evaluating the lower bound when the remaining candidates in set_{can} are very few. N_k records the actual number of loops of k run in the rejection step. In our experiments, $Cond_{Ter}$ is used for PWHT, PGCK and FWHT. $Cond_{Ter}$ is always set as true and *Step a.3* is considered to be skipped for the other algorithms.

The advantage of using $f_{low}(k,j)$ is that it is more efficient to compute $f_{low}(k,j)$ than to compute $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_p^p$ and a small number of iteration k in the rejection step can eliminate a large number of mismatching windows.

The unified framework can be straightforwardly modified to find the window that has the minimum $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_p^p$ among all candidate windows. In such a case, the threshold T is not a constant but represents the best dissimilarity score found so far in the kth loop.

The differences among the algorithms are summarized in Table 3.3. FS, IDA and LRP are applicable for any dissimilarity measure based on the L_p-norm ($p \geq 1$), while recent pattern matching works using FFT, PWHT, PGCK and FWHT are based only on L_2-norm. However, in Section 3.2.3, we show that pattern matching using PWHT, PGCK and FWHT can be based on L_p-norm for $p \geq 1$. FS and FFT include only *Step a* while the others include both *Step a* and *Step b*. The N_{Maxk} is not applicable for FS and FFT; the IDA algorithm has $N_{Maxk} = N_P$; the LRP algorithm has $N_{Maxk} = \log_h N$; PWHT, PGCK and FWHT have $N_{Maxk} = N$. $Cond_{Ter} = Per_{can}^{(k+1)} < \epsilon$ is used for PWHT, PGCK and FWHT.

The FS-equivalent algorithms considered in the chapter achieve high efficiency by using a lower bounding function, $f_{low}(k,j)$, that eliminates mismatching candidates early. Hence, the methods for estimating these lower bounds accurately affect the FS-equivalent algorithms. In the following, we recall the lower bound estimation methods used by IDA, LRP, PWHT, PGCK and FWHT.

3.2.1 The Lower Bounding Function for IDA

The lower bound used by IDA is

$$f_{low}(k,j) = \sum_{m=1}^{k-1} ||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_{p,P_m}^p + \sum_{n=k}^{N_P} |\, ||\vec{\mathbf{x}}_t^{(N)}||_{p,P_n} - ||\vec{\mathbf{x}}_w^{(N,j)}||_{p,P_n}|^p, \qquad (3.1)$$

Methods	FS	FFT [53]	IDA [31]	LRP [30]	PKs [1; 32; 33]
Norm	L_p	L_2	L_p	L_p	L_2
Steps	b		a-b		
N_{Max}	N/A		N_P	$\log_h N$	N
$Cond_{Ter}$	N/A		N/A	N/A	$Per_{can}^{(k+1)} < \epsilon$
f_{low}	N/A		Eqn(3.1)	$\|\|\mathbf{V}^{(u(k)\times N)}(\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)})\|\|_p^p$	

Table 3.3: *Difference for algorithms in the unified framework in Table 3.2. PWHT, PGCK and FWHT share the same column 'PKs'.*

where the left term is the partial L_p-dissimilarity between $\vec{\mathbf{x}}_t^{(N)}$ and $\vec{\mathbf{x}}_w^{(N,j)}$ and the right term is an estimation of the remaining L_p-dissimilarity. Let us denote the modulo operation by %. When partitioning a set of size N into N_P subsets, we constrain that $N\%N_P = 0$ and the subsets P_n for $n = 1, \ldots, N_P$ have the same size, i.e. N/N_P. Thus the candidate window is equally partitioned into N_P subwindows having the same size. The method presented in [31] does not imply this constraint, but this constraint normally makes IDA computationally more efficient and facilitates computational complexity analysis. Thus the constraint is used in this chapter, as it is also the case of the experimental results reported in [31].

3.2.2 The Lower Bounding Function for LRP, PWHT, GCK and FWHT

Algorithms WHT, GCK, FWHT and LRP make use of the following lower bounding function:

$$f_{low}(k,j) = \frac{||\mathbf{V}^{(u(k)\times N)}(\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)})||_p^p}{||\mathbf{V}^{(u(k)\times N)}||_p^p}, \quad (3.2)$$

$$\text{where } \mathbf{V}^{(u(k)\times N)} = [\vec{\mathbf{v}}^{(0)}, \cdots, \vec{\mathbf{v}}^{(i)}, \cdots, \vec{\mathbf{v}}^{(u(k)-1)}]^T,$$

$\vec{\mathbf{v}}^{(i)^T}$ for $i = 0, 1, \cdots, u(k) - 1$ is the ith row of the matrix $\mathbf{V}^{(u(k)\times N)}$, $\vec{\mathbf{v}}^{(i)}$ is the ith basis vector in $\mathbf{V}^{(u(k)\times N)}$, and $u(k)$ is the number of basis vectors chosen for transform domain pattern matching in the kth loop of the rejection step.

LRP

The LRP algorithm is best explained using Kronecker product which is denoted as $\otimes$. If $\mathbf{A}$ is a $U_1 \times Q_1$ matrix (a_{n_1,n_2}) and $\mathbf{B}$ is a $U_2 \times Q_2$ matrix (b_{m_1,m_2}), then $\mathbf{A} \otimes \mathbf{B}$

is the following $U_1U_2 \times Q_1Q_2$ matrix:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{0,0}\mathbf{B} & a_{0,1}\mathbf{B} & \cdots & a_{0,Q_1-1}\mathbf{B} \\ a_{1,0}\mathbf{B} & a_{1,1}\mathbf{B} & \cdots & a_{1,Q_1-1}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{U_1-1,0}\mathbf{B} & a_{U_1-1,1}\mathbf{B} & \cdots & a_{U_1-1,Q_1-1}\mathbf{B} \end{bmatrix}. \tag{3.3}$$

Denote the $1 \times (N/u)$ matrix with all elements equal to 0 and 1 as $\mathbf{0}_{1\times(N/u)}$ and $\mathbf{1}_{1\times(N/u)}$ respectively. The transform matrix for the LRP algorithm proposed in [30] is given by:

$$\begin{aligned} \mathbf{V}^{(u\times N)} &= \mathbf{I}_u \otimes \mathbf{1}_{1\times(N/u)} \\ &= \begin{bmatrix} \mathbf{1}_{1\times(N/u)} & \mathbf{0}_{1\times(N/u)} & \cdots & \mathbf{0}_{1\times(N/u)} \\ \mathbf{0}_{1\times(N/u)} & \mathbf{1}_{1\times(N/u)} & \cdots & \mathbf{0}_{1\times(N/u)} \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{0}_{1\times(N/u)} & \mathbf{0}_{1\times(N/u)} & \cdots & \mathbf{1}_{1\times(N/u)} \end{bmatrix}_{u\times u} \end{aligned} \tag{3.4}$$

where $||\mathbf{I}_u \otimes \mathbf{1}_{1\times(N/u)}||_p^p = (N/u)^{(p-1)/p}$. For example, when $u = 2, N = 4$, we have:

$$\mathbf{V}^{(2\times4)} = \mathbf{I}_2 \otimes \mathbf{1}_{1\times2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes [1\ 1] = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}. \tag{3.5}$$

The LRP matrix in (3.4) contains only 0s and 1s. Thus the projection of input data onto LRP basis vectors requires only additions. For candidate windows in the images, transformation using LRP is computed by summing pixel values in a rectangle. As for the lower bound in (3.2), the $\mathbf{V}^{(u(k)\times N)}$ in (3.2) is given by the $\mathbf{V}^{(u\times N)}$ in (3.4), where $u(k) = u = h^{k-1}$ and h can be any small integer number. The experiments in [30] use $h = 4$.

PWHT, PGCK and FWHT

Let $\mathbf{M}^{(N)} = [\vec{\mathbf{m}}^{(N,0)}, \cdots, \vec{\mathbf{m}}^{(N,N-1)}]^T$ be the $N \times N$ WHT matrix. The 4×4 WHT matrix is as follows:

$$\mathbf{M}^{(4)} = \begin{bmatrix} \vec{\mathbf{m}}^{(4,0)^T} \\ \vec{\mathbf{m}}^{(4,1)^T} \\ \vec{\mathbf{m}}^{(4,2)^T} \\ \vec{\mathbf{m}}^{(4,3)^T} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}. \tag{3.6}$$

Since the elements in WHT basis vectors contain only 1 or -1, projection of input data onto WHT basis vectors requires only additions and subtractions. When $N = 8$, the 8 row vectors in $\mathbf{M}^{(8)}$ are shown in Figure 3.1. The WHT in (3.6) and in Figure 3.1 are in sequency order. For sequency-ordered WHT, the spatial frequency extracted by the basis vector increases as the index i of basis vectors $\vec{\mathbf{m}}^{(N,i)}$ increases. We have $||\mathbf{M}^{(u(k)\times N)}||_2^2 = N$ for WHT matrix.

The transform matrix for the GCK in [32] can be represented as follows:

$$\mathbf{V}^{(N)} = \mathbf{M}^{(N/R)} \otimes \mathbf{S}^{(R)}, \tag{3.7}$$

where $\mathbf{M}^{(N/R)}$ is the $N/R \times N/R$ WHT matrix and $\mathbf{S}^{(R)}$ can be any $R \times R$ orthogonal matrix. We have $||\mathbf{V}^{(u(k)\times N)}||_2^2 = N/R$ for the GCK matrix in (3.7).

Considering the general lower bounding function defined in (3.2), we can select basis vectors of the $\mathbf{V}^{u(k)\times N}$ in (3.2) from the LRP matrix in (3.4), from the WHT matrix $\mathbf{M}^{(N)}$ or from the GCK matrix in (3.7). For example, if $N = 4$, $k = 2$ and $u(k) = k = 2$, then we can select the first 2 WHT basis vectors $\vec{\mathbf{m}}^{(4,0)}$ and $\vec{\mathbf{m}}^{(4,1)}$ in (3.6) for constructing the matrix $\mathbf{V}^{(2\times 4)} = [\vec{\mathbf{m}}^{(4,0)}\ \vec{\mathbf{m}}^{(4,1)}]^T$ and use this matrix as the $\mathbf{V}^{(u(k)\times N)}$ in (3.2).

3.2.3 Further Analysis on Pattern Matching using Transformation

The definition of induced matrix p-norms is as follows:

$$||\mathbf{V}^{(u(k)\times N)}||_p = \max_{\vec{\mathbf{z}}\neq 0} \frac{||\mathbf{V}^{(u(k)\times N)}\vec{\mathbf{z}}||_p}{||\vec{\mathbf{z}}||_p}. \tag{3.8}$$

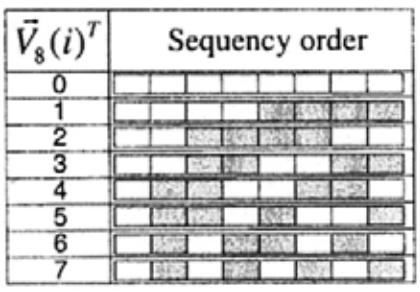


Figure 3.1: *WHT basis vectors in sequency order. White represents the value +1 and grey represents the value −1.*

And from (3.8) it follows that when $\mathbf{V}^{(u(k)\times N)} \neq 0$:

$$||\vec{\mathbf{z}}||_p \geq \frac{||\mathbf{V}^{(u(k)\times N)}\vec{\mathbf{z}}||_p}{||\mathbf{V}^{(u(k)\times N)}||_p}. \tag{3.9}$$

As pointed out in [30], the lower bound in (3.2) is derived from (3.9) by replacing the $\vec{\mathbf{z}}$ in (3.9) with $\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}$:

$$||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_p^p \geq \frac{||\mathbf{V}^{(u(k)\times N)}(\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)})||_p^p}{||\mathbf{V}^{(u(k)\times N)}||_p^p} = f_{low}(k,j). \tag{3.10}$$

The following inequality about L_2 norm is shown in [1]:

$$\begin{aligned} &||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_2^2 = ||\vec{\mathbf{d}}_{t,w}^{(j)}||_2^2 \\ &\geq (\mathbf{V}^{(u(k)\times N)}\vec{\mathbf{d}}_{t,w}^{(j)})^T(\mathbf{V}^{(u(k)\times N)}\mathbf{V}^{(u(k)\times N)^T})^{-1}(\mathbf{V}^{(u(k)\times N)}\vec{\mathbf{d}}_{t,w}^{(j)}) \\ &= f_{low}(k,j). \end{aligned} \tag{3.11}$$

where $\vec{\mathbf{d}}_{t,w}^{(j)} = \vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}$. The $f_{low}(k,j)$ in (3.10) is used for LRP while the $f_{low}(k,j)$ in (3.11) is used for PWHT, PGCK and FWHT. Let us now consider the L_2 norm, if the basis vectors in $\mathbf{V}^{(u(k)\times N)}$ in (3.11) are orthonormal, then we have $||\mathbf{V}^{(u(k)\times N)}||_2 = 1$ in (3.10) and $(\mathbf{V}^{(u(k)\times N)}\mathbf{V}^{(u(k)\times N)^T})^{-1} = \mathbf{I}_{u(k)}$ in (3.11). Under this condition, both (3.10) and (3.11) yield the following relation:

$$||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_2^2 \geq ||\mathbf{V}^{(u(k)\times N)}(\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)})||_2^2 = f_{low}(k,j). \tag{3.12}$$

The inequality in (3.10) is more general than that in (3.11) because: 1) the inequality in (3.11) is only applicable for L_2 norm while the inequality in (3.10) is applicable for L_p norm for $p \geq 1$; 2) $\mathbf{V}^{(u(k)\times N)}$ is required to have rank $u(k)$ for the inverse matrix in (3.11) while this is not required in (3.10).

The following theorem describes the relationship between LRP and WHT:

Theorem 3.1 When the $u = 2^n$ basis vectors in $\mathbf{V}_{WHT}^{(u\times N)}$ are the first sequency-ordered WHT basis vectors, we have the LRP transform matrix:

$$\mathbf{V}_{LRP}^{(u\times N)} = \mathbf{I}_u \otimes \mathbf{1}_{1\times(N/u)}, \tag{3.13}$$

such that: 1) the subspace spanned by the u basis vectors in $\mathbf{V}_{WHT}^{(u\times N)}$ is equal to the subspace spanned by the u basis vectors in $\mathbf{V}_{LRP}^{(u\times N)}$; 2) for any length-$N$ input vector

$\vec{\mathbf{x}}$, if the basis vectors in $\mathbf{V}_{LRP}^{(u\times N)}$ and $\mathbf{V}_{WHT}^{(u\times N)}$ are normalized to have L_2 norm equal to 1, then $||\mathbf{V}_{WHT}^{(u\times N)}\vec{\mathbf{x}}||_2^2 = ||\mathbf{V}_{LRP}^{(u\times N)}\vec{\mathbf{x}}||_2^2$; 3) the transformation $\mathbf{V}_{WHT}^{(u\times N)}\vec{\mathbf{x}}$ requires at least $3u/2$ additions per pixel while the transformation $\mathbf{V}_{LRP}^{(u\times N)}\vec{\mathbf{x}}$ requires 3 additions per pixel when computed on sliding windows.

The proof for Theorem 3.1 is provided in the appendix. The more is the energy packed, the more are candidates can be eliminated by the rejection step. Theorem 3.1 states that the $u = 2^n$ sequency ordered basis vectors in $\mathbf{V}_{WHT}^{(u\times N)}$ and the u basis vectors in $\mathbf{V}_{LRP}^{(u\times N)} = \mathbf{I}_u \otimes \mathbf{1}_{1\times(N/u)}$ are the same in subspace spanning and energy packing ability. However, the larger is u, the more computationally efficient is the transformation $\mathbf{V}_{LRP}^{(u\times N)}\vec{\mathbf{x}}$ than $\mathbf{V}_{WHT}^{(u\times N)}\vec{\mathbf{x}}$.

The algorithms PWHT, PGCK and FWHT were proposed for the L_2 norm only. Recently, WHT have been used for motion estimation in [2; 9] based on L_1 norm, but the algorithms are not FS-equivalent. Actually, we can see from the analysis in (3.10) that the basis vectors selected from WHT and GCK are also applicable for FS-equivalent pattern matching using L_p norms for $p \geq 1$ as dissimilarity measure.

3.3 Computational Analysis of Algorithms IDA, PWHT, PGCK, FWHT and LRP

Execution time was used for assessing the computational efficiency of IDA, PWHT, PGCK and FWHT in [1; 31–33]. But literature lacks a computational analysis of these algorithms. Since execution time varies with hardware, the execution time measurements in [1; 31–33] cannot provide an accurate comparison of the four algorithms. In this section we analyze the computational complexity of IDA, PWHT, PGCK, FWHT and LRP by counting the number of operations required by each algorithm. The number of operations required by the algorithms is summarized in Table 3.4, those required in the rejection step and FS-step are listed separately in Table 3.5.

To recall the notation already introduced, the $N_1 \times N_2$ pattern has $N = N_1 N_2$ pixels, the $J_1 \times J_2$ image has $J = J_1 J_2$ pixels and W candidate windows. At each iteration of k for $k = 1, 2, \ldots$ in the rejection step, some candidate windows are eliminated and the remaining ones will be considered in the next loop. We denote the number of candidates examined at iteration k in *Step a* as $N_{can}^{(k)}$ and the number of candidates examined in the FS-step as $N_{can}^{(FS)}$. Initially, $k = 1$, all candidates are examined and

we have $N_{can}^{(1)} = W$.

In the computational analysis, subtractions are given the same weight as additions. The computation of certain terms depends on the computation of the L_p norm in (1.1), which in turn is related to the computation of $|z|^p$ and $\sqrt[p]{z}$. The operations for computing $|z|^p$ and $\sqrt[p]{z}$ are called here power-p operation and root-p operation respectively. When $p = 2$, power-p operation computes the square of z, i.e. z^2, and root-p operation computes the square root of z, i.e. $\sqrt{z}$. When $p = 1$, power-p operation computes absolute value of z, i.e. $|z|$, and root-p operation does nothing.

The terms related to the pattern $\vec{\mathbf{x}}_t^{(N)}$ are computed once and for all at initialization time and hence not considered in the computational complexity since the associated complexity is negligible compared with the other computations. As a common step for algorithms IDA, PWHT, PGCK, FWHT and LRP, *Step a.2* of Table 3.2 checks condition $f_{low}(k,j) > T$ for the $N_{can}^{(k)}$ candidates at iteration k, which requires $\sum_k N_{can}^{(k)}$ comparison operations. The computation required by *Step a.3* of Table 3.2 is negligible. The remaining steps in Table 3.2 that require analysis are *Step a.1* of the Rejection step, which computes the lower bounding function $f_{low}(k,j)$, and the FS-step. In the following, we analyze the computational complexity of these two steps for each of the considered algorithms.

3.3.1 Computational Analysis for IDA

The computation of the lower bounding function for IDA is illustrated and analyzed in Table 3.6. When $k = 1$, which is *Case 1* of Table 3.6, the lower bounding function in (3.1) is given by:

$$f_{low}(1,j) = \sum_{m=1}^{N_P} | \, ||\vec{\mathbf{x}}_t^{(N)}||_{p,P_m} - ||\vec{\mathbf{x}}_w^{(N,j)}||_{p,P_m}|^p. \tag{3.14}$$

When $k > 1$, i.e. *Case 2* of Table 3.6, the lower bounding function is:

$$\begin{aligned} f_{low}(k,j) = f_{low}(k-1,j) &+ ||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||_{p,P_k}^p \\ &- | \, ||\vec{\mathbf{x}}_t^{(N)}||_{p,P_k} - ||\vec{\mathbf{x}}_w^{(N,j)}||_{p,P_k}|^p. \end{aligned} \tag{3.15}$$

As a summary of Table 3.6, IDA requires $4W + 2N_PW + \sum_{k=2}^{N_P-1}[(\frac{2N}{N_P} + 2)N_{can}^{(k)}]$ additions, $J + N_PW + \frac{N}{N_P}(\sum_{k=2}^{N_P-1} N_{can}^{(k)})$ power-p operations and W root-p operations

Methods	Number of additions
FS	$2NW$
FFT	$6Jlog_2J + 7W$
IDA	$4W + 2N_PW + \sum_{k=2}^{N_P-1}[(\frac{2N}{N_P}+2)N_{can}^{(k)}] + \frac{2N}{N_P}N_{can}^{(FS)}$
LRP_{sld}	$2J + \sum_k (2A_{LRP}^{(k,h)} + 3W) + 2NN_{can}^{(FS)}$
LRP_{can}	$2J + \sum_k (2A_{LRP}^{(k,h)} + 3A_{LRP}^{(k,h)}) + 2NN_{can}^{(FS)}$
PKs	$B(N, N_k, W) + 2\sum_k N_{can}^{(k)} + 2NN_{can}^{(FS)}$
Methods	Number of power-p operations
FS	NW
FFT	$6Jlog_2J + J$
IDA	$J + N_PW + \frac{N}{N_P}(\sum_{k=2}^{N_P-1} N_{can}^{(k)}) + \frac{N}{N_P}N_{can}^{(FS)}$
LRP	$\sum_k A_{LRP}^{(k,h)} + NN_{can}^{(FS)}$
PKs	$\sum_k N_{can}^{(k)} + NN_{can}^{(FS)}$
Methods	Number of root-p operations
IDA	W
Methods	Number of comparison operations (in Step a.2)
	$\sum_k N_{can}^{(k)}$

Table 3.4: *Number of operations required by the algorithms. PWHT, PGCK and FWHT share the same row 'PKs'. LRP_{sld} computes the transformation in a sliding manner for all pixel locations; LRP_{can} computes the transformation for $N_{can}^{(k)}$ candidate windows at iteration k. LRP_{sld} and LRP_{can} share the same row "LRP"' for the number of power-p operations. In this table, $A_{LRP}^{(k,h)} = h^{(k-1)}N_{can}^{(k)}$. The number of comparison operations is applicable to IDA, PWHT, PGCK, FWHT and LRP.*

for obtaining the lower bounding function.

In the FS-step, IDA computes only the L_p norm dissimilarity for pixels in the last subwindow restricted by partition P_{N_P}. This subwindow contains $\frac{N}{N_P}$ pixels for each candidate window. Thus, the FS-step requires $\frac{2NN_{can}^{(FS)}}{N_P}$ additions and $\frac{NN_{can}^{(FS)}}{N_P}$ power-p operations.

3.3.2 Computational Analysis for LRP

As pointed out in [30], LRP requires $\sum_k [2A_{LRP}^{(k,h)} + (h-1)J]$ additions and $\sum_k (A_{LRP}^{(k,h)})$ power-p operations for obtaining the lower bounding function, where $A_{LRP}^{k,h} = h^{(k-1)}N_{can}^{(k)}$. In the FS-step, LRP requires $2NN_{can}^{(FS)}$ additions and $NN_{can}^{(FS)}$ power-p operations.

Methods	Number of additions	
	Rejection Step	FS-step
IDA	$4W+2N_PW+\sum_{k=2}^{N_P-1}[(\frac{2N}{N_P}+2)N_{can}^{(k)}]$	$\frac{2N}{N_P}N_{can}^{(FS)}$
LRP_{sld}	$2J+\sum_k(2A_{LRP}^{(k,h)}+3W)$	$2NN_{can}^{(FS)}$
LRP_{can}	$2J+\sum_k(2A_{LRP}^{(k,h)}+3A_{LRP}^{(k,h)})$	
PKs	$B(N,N_k,W)+2\sum_k N_{can}^{(k)}$	
Methods	Number of power-p operations	
	Rejection Step	FS-step
IDA	$J+N_PW+\frac{N}{N_P}\sum_k N_{can}^{(k)}$	$\frac{N}{N_P}N_{can}^{(FS)}$
LRP	$\sum_k A_{LRP}^{(k,h)}$	$NN_{can}^{(FS)}$
PKs	$\sum_k N_{can}^{(k)}$	

Table 3.5: *Numbers of additions and power-p operations required by the algorithms in the Rejection and FS steps. PWHT, PGCK and FWHT share the same row 'PKs'. LRP_{sld} and LRP_{can} share the same row "LRP"' for the number of power-p operations.*

The sliding transformation $\mathbf{V}^{(u(k)\times N)}\vec{\mathbf{x}}_w^{(N,j)}$ in $f_{low}(k,j)=\frac{||\mathbf{V}^{(u(k)\times N)}\mathbf{x}_t^{(N)}-\mathbf{V}^{(u(k)\times N)}\mathbf{x}_w^{(N,j)}||_p^p}{||\mathbf{V}^{(u(k)\times N)}||_p^p}$ is actually the sum of pixel values in rectangles when LRP is used for images. This transformation is computed by hierarchical summation on the entire image in [30], which requires $(h-1)J$ additions at each iteration k.

Alternatively, the integral image method initially requires about $2J$ additions for constructing the integral image and then $3W$ additions for each iteration k for computing the sliding transformation over the image. This approach is denoted as LRP_{sld}. LRP_{sld} is advantageous over the hierarchical summation proposed in [30] because the computation required by LRP_{sld} is independent of h.

As another alternative way, we may compute the transformation for the $N_{can}^{(k)}$ candidates instead of computing it for the entire image, which would require $3h^{(k-1)}N_{can}^{(k)}$ additions for each k using the integral image method. This approach is denoted as LRP_{can}.

Case 1: $k = 1$:
a.1.1: for each pixel $x_{w,a}$ in the image: {Obtain $\|x_{w,a}\|^p$}. ** J power-p operations are required for J pixels. **a.1.2**: for j=0 to $W-1$ { for m=1 to N_P {Obtain $\|\vec{\mathbf{x}}_w^{(N,j)}\|_{p,P_m}$} }. ** $\|\vec{\mathbf{x}}_w^{(N,j)}\|_{p,P_m} = \left(\sum_{n\in P_m} \|x_{w,j,n}\|^p\right)^{\frac{1}{p}}$ as defined in (1.4) is obtained by two steps: 1) sum up $\|x_{w,j,n}\|^p$ in the rectangle restricted by $\|\vec{\mathbf{x}}_w^{(N,j)}\|_{p,P_m}$, where $\|x_{w,j,n}\|^p$ has been obtained as $\|x_{w,a}\|^p$ in the previous step; 2) obtain the pth root of the sum. $4W$ additions are required for obtaining the sum in the W rectangles using the box filtering technique [47]. W root-p operations are required for obtaining the pth root of the W sums. **a.1.3**: for j=0 to $W-1$ { $f_{low}(1,j) = \sum_{n=0}^{N_P-1} \| \|\vec{\mathbf{x}}_t^{(N)}\|_{p,P_n} - \|\vec{\mathbf{x}}_w^{(N,j)}\|_{p,P_n}\|^p$ }. ** $N_P W$ power-p operations, $2N_P W$ additions are required.
Case 2: $k > 1$:

a.1.4: for $N_{can}^{(k)}$ candidates $\vec{\mathbf{x}}_w^{(N,j)}$ in set_{can}:
{Compute $\|\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}\|_{p,P_k}^p$.}
** $\|\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}\|_{p,P_k}^p$ as defined in (1.6) is the L_p norm of length-$\frac{N}{N_P}$ vector. $\frac{2N_{can}^{(k)}N}{N_P}$ additions and $\frac{N_{can}^{(k)}N}{N_P}$ power-p operations are required at iteration k.
a.1.5: for $N_{can}^{(k)}$ candidates $\vec{\mathbf{x}}_w^{(N,j)}$ in set_{can}: {
$f_{low}(k,j) = f_{low}(k-1,j) + \|\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}\|_{p,P_k}^p - \| \|\vec{\mathbf{x}}_t^{(N)}\|_{p,P_k} - \|\vec{\mathbf{x}}_w^{(N,j)}\|_{p,P_k}\|^p$.
}
** $2N_{can}^{(k)}$ additions are required at iteration k for summing up the three terms that have been computed previously.

Table 3.6: *Procedure and corresponding number of operations required by IDA for computing the lower bounding function.*

3.3.3 Analysis for PWHT, PGCK and FWHT

The lower bounding function for PWHT, PGCK and FWHT is computed as:

$$f_{low}(k,j) = f_{low}(k-1,j) + |\vec{\mathbf{v}}^{(k-1)^T}\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{v}}^{(k-1)^T}\vec{\mathbf{x}}_w^{(N,j)}|^p, \tag{3.16}$$

where $k = 1, 2, \ldots, N_k$, $f_{low}(0,j) = 0$. N_k records the actual number of iterations run in the rejection step. The procedure and number of operations required to obtain the lower bounding function is reported in Table 3.7. Let $B(N, N_k, W)$ be the number of operations required by PWHT, PGCK or FWHT to obtain $\mathbf{V}^{(N_k \times N)}\vec{\mathbf{x}}_w^{(N,j)}$ for W candidates. PWHT [1] has two different approaches to compute the transformation: top-down and bottom-up. We have $B(N, N_k, W) = W N_k \log N$ in the worst

a.1.1: for $N_{can}^{(k)}$ windows $\vec{\mathbf{x}}_w^{(N,j)}$ in set_{can}:
{ Obtain $\vec{\mathbf{v}}^{(k-1)^T}\vec{\mathbf{x}}_w^{(N,j)}$ }.
** $B(N, N_k, W)$ additions are required for obtaining $\mathbf{V}^{(N_k \times N)}\vec{\mathbf{x}}_w^{(N,j)}$ for all candidates, where $\vec{\mathbf{v}}^{(k-1)^T}\vec{\mathbf{x}}_w^{(N,j)}$ is one of its elements.
a.1.2: for $N_{can}^{(k)}$ windows $\vec{\mathbf{x}}_w^{(N,j)}$ in set_{can}:
{ Obtain $\lvert\vec{\mathbf{v}}^{(k-1)^T}\vec{\mathbf{x}}_l^{(N)} - \vec{\mathbf{v}}^{(k-1)^T}\vec{\mathbf{x}}_w^{(N,j)}\rvert^p$ }.
** $N_{can}^{(k)}$ additions and $N_{can}^{(k)}$ power-p operations are required at iteration k.
a.1.3: for $N_{can}^{(k)}$ windows $\vec{\mathbf{x}}_w^{(N,j)}$ in set_{can}:
$\{f_{low}(k,j) = f_{low}(k-1,j) + \lvert\vec{\mathbf{v}}^{(k-1)^T}\vec{\mathbf{x}}_l^{(N)} - \vec{\mathbf{v}}^{(k-1)^T}\vec{\mathbf{x}}_w^{(N,j)}\rvert^p$ }.
** $N_{can}^{(k)}$ additions are required at iteration k for summing up the two terms that have been computed previously.

Table 3.7: *Procedure and corresponding computation required by PWHT, GCK and FWHT for computing the lower bounding function.*

case and $B(N, N_k, W) = 2WN_k$ in the best case for top-down PWHT. For bottom-up PWHT, $B(N, N_k, W) = \sum_k NN_{can}^{(k)}$ in the worst case and $B(N, N_k, W) = \sum_k N_{can}^{(k)}$ in the best case. As for PGCK and FWHT, $B(N, N_k, W) = 2WN_k$ and $B(N, N_k, W) = 3WN_k/2$ respectively. As a summary of Table 3.7, PWHT, PGCK and FWHT need $B(N, N_k, W) + 2\sum_k N_{can}^{(k)}$ additions and $\sum_k N_{can}^{(k)}$ power-p operations in order to calculate the lower bounding function.

In the FS-step, $2NN_{can}^{(FS)}$ additions and $NN_{can}^{(FS)}$ power-p operations are needed by PWHT, PGCK and FWHT.

3.3.4 New Termination Condition

As illustrated before, the termination condition in [1] has been used by PWHT and PGCK for early termination of the rejection step, which corresponds to the $Cond_{Ter}$ at *Step a.3* in Table 3.2. Let the computation required for the FS-step be C_{FS} and the computation required for iteration $k+1$ in the rejection step be $C_{Rej}^{(k+1)}$. The idea behind the strategy in [1] is that the rejection step should be terminated when it is more efficient to directly calculate the actual distance of the remaining candidates in the FS-step than to continue the computation in the rejection step, i.e. when $C_{FS} < C_{Rej}^{(k+1)}$. The strategy proposed in [1] terminates the rejection step when the percentage of remaining candidate windows checked in iteration $k+1$, i.e. $Per_{can}^{(k+1)}$, is below a certain threshold ϵ. Hence, in the absence of a computational analysis, setting a threshold ϵ is intuitively a simplified version of the comparison $C_{FS} < C_{Rej}^{(k+1)}$. However, the computational

analysis carried out in Section 3.3.3 allows us to devise a novel and principled approach to the early termination strategy.

The computation required in iteration $k+1$ of the rejection step is:

$$C_{Rej}^{(k+1)} = \{B(N,k+1,W) - B(N,k,W) + 2N_{can}^{(k+1)} \text{ add}, \\ N_{can}^{(k+1)} \text{ power-}p, N_{can}^{(k+1)} \text{ cmp}\}, \tag{3.17}$$

where "add" stands for additions, "power-p" stands for power-p operations and "cmp" stands for comparison operations. If we terminate the rejection step before entering iteration $k+1$, the computation required in the FS-Step is:

$$C_{FS} = \{2NN_{can}^{(k+1)} \text{ add}, NN_{can}^{(k+1)} \text{ power-}p\}. \tag{3.18}$$

Thus, the first termination condition, which is denoted as $Cond_{Ter,1}$, is true when $C_{Rej}^{(k+1)} > C_{FS}$, i.e. when the computational complexity of iteration $k+1$ of the rejection step is greater than the computational complexity of the forthcoming FS-step.

At iterations k and $k+1$ of the rejection step, there are respectively $N_{can}^{(k)}$ and $N_{can}^{(k+1)}$ candidates undergoing the rejection scheme. On one hand, the kth iteration of the rejection step eliminates $N_{can}^{(k)} - N_{can}^{(k+1)}$ candidates from the set of candidates and thus saves the computation required in FS-step for these $N_{can}^{(k)} - N_{can}^{(k+1)}$ candidates; on the other hand, iteration k of rejection step itself needs computation. The second termination condition, which is denoted as $Cond_{Ter,2}$, is true when the computation required by the kth iteration in the rejection step is greater than the computation saved by excluding $N_{can}^{(k)} - N_{can}^{(k+1)}$ candidates from calculating their actual distances in the FS-step.

As a simple example of $Cond_{Ter,2}$, when $N_{can}^{(k)} = N_{can}^{(k+1)} = N_{can}^{(k+2)} = ...$, we have no computation saved by the rejection step. Thus $Cond_{Ter,2}$ is true in this case. In this example, the rejection step should be terminated because it cannot efficiently reject any candidates although it requires some amount of computation.

In summary, the new termination strategy based on computational analysis consists in terminating the rejection step if $Cond_{Ter,1}$ is true or $Cond_{Ter,2}$ is true. The former is true when the computation required for the iteration $k+1$ of the rejection step is greater than the computation required for the FS-step. The latter is true when the computation required by iteration k of the rejection step is greater than the computation it saves. In

Dataset	Image size	J	Pattern size	N
$Scale1-1$	160×120	19200	16×16	256
$Scale2-1$	320×240	76800	16×16	256
$Scale2-2$	320×240	76800	32×32	1024
$Scale3-1$	640×480	307200	16×16	256
$Scale3-2$	640×480	307200	32×32	1024
$Scale3-3$	640×480	307200	64×64	4096
$Scale4-1$	1280×960	1228800	16×16	256
$Scale4-2$	1280×960	1228800	32×32	1024
$Scale4-3$	1280×960	1228800	64×64	4096
$Scale4-4$	1280×960	1228800	128×128	16384
$Scale5-1$	2560×1920	4915200	16×16	256
$Scale5-2$	2560×1920	4915200	32×32	1024
$Scale5-3$	2560×1920	4915200	64×64	4096
$Scale5-4$	2560×1920	4915200	128×128	16384

Table 3.8: *Datasets used in the experiments.*

the following experiments, this new termination strategy is used for PGCK and FWHT unless specified otherwise. We will also compare the original strategy in [1] with the proposed strategy in Section 3.5.8.

3.4 Performance Evaluation

3.4.1 Dataset

In order to evaluate the performance of the compared algorithms, 14 datasets containing different sizes of patterns and images are used. As shown in Table 3.8, the datasets are denoted as $Scalen_1 - n_2$ for $n_1 = 1, 2, 3, 4, 5, n_2 = 1, \ldots, 4$, where n_1 corresponds to image size and n_2 corresponds to pattern size. For example, datasets $Scale2-1$ and $Scale2-2$ have the same image size 320×240 but different pattern sizes.

The experiments include a total of 150 greyscale images chosen among three databases: MIT [54], medical [55], and remote sensing [56]. 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 150 images have been subdivided into five groups of 30 images, each group being characterized by a size of images in $Scale1-1, Scale2-1, Scale3-1, Scale4-1, Scale5-1$, (i.e. 160×120, 320×240, 640×480, 1280×960, 2560×1920). For each image, 10 patterns were

randomly selected among those showing a standard deviation of pixel intensities higher than a threshold (i.e., 45) for each dataset. Datasets having the same image size share the same images but have different patterns in both size and location. For example, datasets $Scale2-1$ and $Scale2-2$ share the same 30 images having size 320×240 but have different patterns in size and location. So each dataset contains 300 image-pattern pairs. Since we have 14 datasets, there are 4200 image-pattern pairs in all. This dataset is originated from that in [31] and extended in this chapter to include more sizes of patterns.

3.4.2 Evaluation Criterion

In the experiments, we will examine the evaluated algorithms using both SSD and SAD as the dissimilarity measure between pattern and candidate window. Given a pattern having N pixels, the SSD threshold, T_{SSD}, and SAD threshold, T_{SAD}, are set as follows:

$$\begin{aligned} T_{SSD} &= 1.1 \cdot SSD_{min} + N, \\ T_{SAD} &= 1.1 \cdot SAD_{min} + N. \end{aligned} \tag{3.19}$$

where SSD_{min} and SAD_{min} are respectively the SSD and SAD between the pattern and the best matching window. If the SSD(SAD) between any candidate window in the image and the pattern is below the threshold, the candidate window is regarded to match the pattern.

We developed the code for IDA and FWHT since we are authors of these algorithms. The code for Hel-Or and Hel-Or's PWHT [1] is from the authors' website [57], with some small modifications introduced by us to deal with large pattern. The code for PGCK [32] is based on the code provided by Hel-Or and Hel-Or, which was previously used for motion estimation in [2] and was modified by us for pattern matching task. Finally, we wrote the code for LRP according to the algorithm described in [30]. All of the algorithms are written in C, compiled with VC 6.0 and run on windows XP systems as single-thread tasks. The parameters for PWHT use the default values in [57], the parameters for IDA are the same as in [31]. The parameter h for LRP is chosen to be 4, which is the same as in [30].

Since all the evaluated algorithms find the same matching windows as the FS, the only concern is computational efficiency, which is assessed here in terms of both

Environment	CPU	Memory size
*Env*1	Intel core 2 (6400) 2.13GHz	3G
*Env*2	Intel Pentium 4 2.8GHz	1G
*Env*3	Intel Xeon 3GHz	2G
*Env*4	AMD Athlon 64 X2 2.21GHz	3G

Table 3.9: *Hardware environment used in the experiments.*

execution time and required number of operations. In particular, results are measured as speed-ups over the FS algorithm in terms of execution time and number of operations. As an example, the speed-up of IDA over FS in execution time or in number of operation is measured as the execution time or number of operations required by FS divided by that required by IDA. As listed in Table 3.9, we have measured the execution time on 3 Intel CPUs and 1 AMD CPU with different memory sizes. If not differently specified, the reported speed-ups in execution time are averages of the speed-ups measured in each of the 4 environments. As an example, the speed-up of IDA in execution time is the average of the speed-ups of IDA over FS measured in the 4 environments.

In Section 3.3.2, two different implementations of the LRP algorithm were introduced. LRP_{sld} computes the transformation in a sliding manner on the entire image while LRP_{can} computes the transformation for $N_{can}^{(k)}$ candidate windows at loop k. Both implementations were evaluated in the experiments.

3.5 Experimental results

3.5.1 Experiment on Images Without Noise

In this experiment, we evaluate algorithms on the datasets described in Table 3.8, which correspond to different sizes of images and patterns.

The speed-ups in execution time yielded by the evaluated algorithms using SSD as the dissimilarity measure are shown in Figure 3.2. In this experiment, the algorithms can be ordered from fastest to slowest as follows: 1) PGCK, 2) FWHT, IDA, and LRP_{can}, 3) LRP_{sld}, 4) PWHT, 5) FFT. PGCK is faster than FWHT because very few (less than 4) basis vectors are computed in this experiment. The experimental results in [33] also show that FWHT is slower than PGCK when the number of basis vectors used is less than 4. With the exception of dataset $Scale4-4$, where FFT is faster than PWHT, FFT is always slower than the other evaluated algorithms.

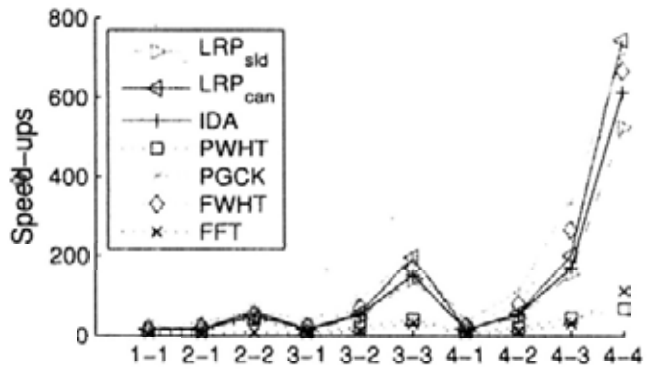


Figure 3.2: *Speed-ups in execution time for images without noise when SSD is used. X-axis corresponds to datasets* $Scale1-1, Scale2-1, \ldots, Scale4-4$ *in Table 3.8.*

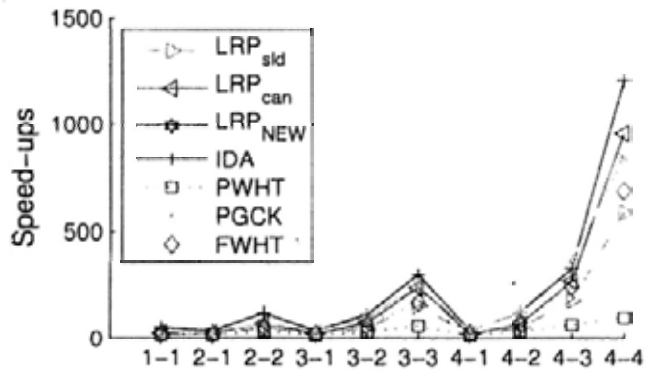


Figure 3.3: *Speed-ups in execution time for images without noise when SAD is used. X-axis corresponds to datasets* $Scale1-1, Scale2-1, \ldots, Scale4-4$ *in Table 3.8.*

The speed-ups in execution time using SAD are shown in Figure 3.3. PGCK and IDA are the fastest in this experiment, with IDA faster than PGCK in datasets $Scale1-1$, $Scale2-1$, $Scale2-2$, $Scale3-2$ and $Scale4-4$. Ordering of the other algorithms is similar to that attained using SSD (Figure 3.2).

3.5.2 Experiment on Images with Gaussian Noise

In this experiment, 4 different levels of iid zero-mean Gaussian noise are added to each image of the datasets described in Table 3.8. The 4 Gaussian noise levels having variances 325, 650, 1300 and 2600 [2] for 8-bit pixel value are referred to as $G(1)$, $G(2)$, $G(3)$ and $G(4)$ respectively. Figure 3.4 shows an image from the dataset in Table 3.8 and its distorted images with noise levels $G(1)$ to $G(4)$, where the 640×480 distorted images have PSNR 23.23, 20.4, 17.7 and 16.1 when compared with the original image.

The speed-ups in execution time using SSD for this experiment are shown in Figure

[2]Corresponding to 0.005, 0.01, 0.02 and 0.03 on normalized pixel intensities ranging within [0, 1].

3.5. Each sub-figure in Figure 3.5 corresponds to a dataset in Table 3.8. In Figure 3.5, the top row corresponds to the smallest image size and bottom row corresponds to the largest image size; the leftmost column corresponds to the smallest pattern size and the rightmost column corresponds to the largest pattern size. This is similar for Figure 3.6 - Figure 3.13.

As shown in Figure 3.5, the speed-up of FFT over FS is independent of the noise level while the speed-ups of the other fast algorithms decrease as the noise level increases from $G(1)$ to $G(4)$, so that FFT turns out to be in most cases the fastest method when the noise level is very high, i.e. $G(4)$ in Figure 3.5. At the lower noise levels LRP_{sld} is in most cases the fastest algorithm, and quite close to the fastest in other cases. FWHT turns out faster than PGCK. This is because in this experiment more than 5 basis vectors are required for rejecting mismatched candidate windows, where FWHT is faster than PGCK in computing the transformation. PGCK is faster than PWHT in most cases because PGCK is more efficient than PWHT in computing the transformation. FWHT is faster than LRP when noise is low and pattern size is small especially when image size is large. When the number of required basis vectors is large, LRP outperforms FWHT in computing the transformation, which is proved in Theorem 3.1. IDA preforms well when pattern-size is 16×16, i.e. $Scale1-1, Scale2-1, Scale3-1, Scale4-1$, and has similar performance as PWHT in other cases.

The speed-ups in number of operations using SSD are shown in Figure 3.6. It is clear that the range of the execution time speed-ups in Figure 3.5 is reduced by 3 to 5 times compared with that of Figure 3.6, hence the speed-ups in execution time for all algorithms but the FFT are significantly worse compared to the respective speed-ups in number of operations. Numerical discrepancies between speed-ups in number of operations and speed-ups in execution time are not surprising for a number of reasons, which include computational analysis weighting equally different kinds of operations, the level of optimization of algorithm implementations, the impact of memory access (e.g. hit rate and miss penalties of cache memories). Nevertheless, the important point here is that, disregarding the Y-axis difference, Figure 3.6 is similar to Figure 3.5, with the ordering of the algorithms being similar in both Figures. In particular, LRP_{sld} is the fastest in most cases, followed by LRP_{can}, FWHT, PGCK, IDA and FFT.

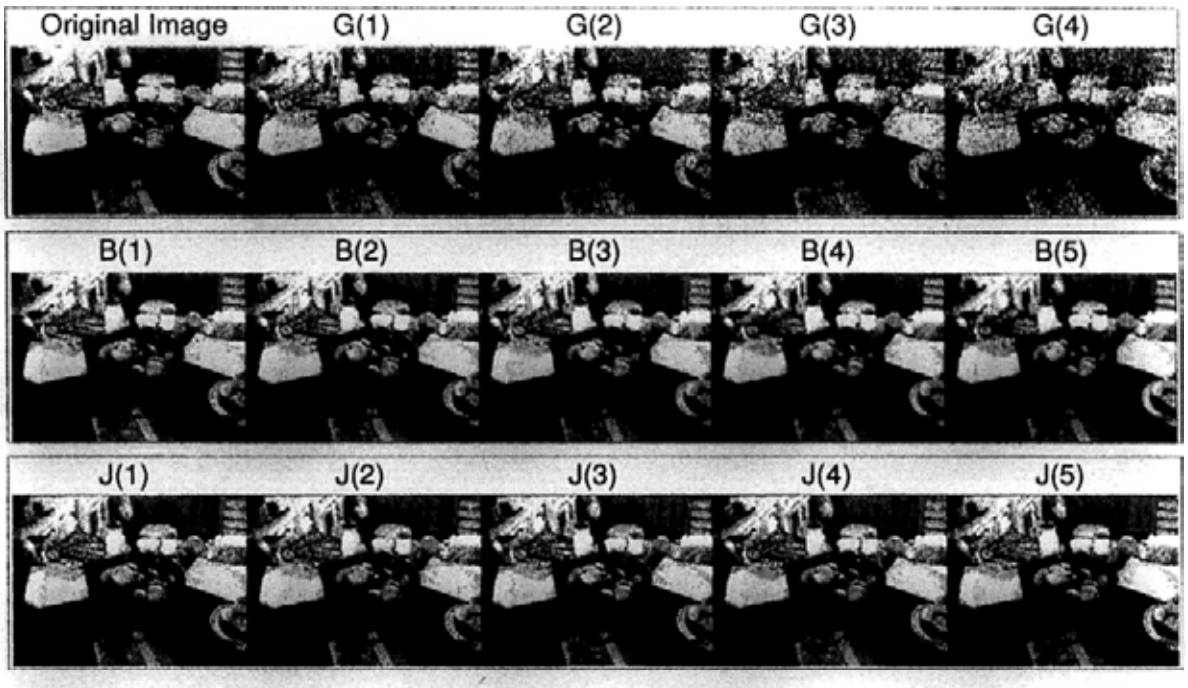


Figure 3.4: *An image from the dataset and its distorted images. 1st row: the original image and its images with Gaussian noise levels $G(1)$ to $G(4)$; 2nd row: images with blurring levels $B(1)$ to $B(5)$; 3rd row: images with JPEG compression quality levels $J(1)$ to $J(5)$.*

3.5.3 Experiment for Blurred Images

In this experiment, 5 different levels of Gaussian low pass filter are used for blurring each image of the datasets described in Table 3.8. The 5 blurring levels, which are referred to as $B(1)$, B(2), $B(3)$, $B(4)$ and $B(5)$, correspond to Gaussian low pass filter having standard deviation $\sigma = 0.2, 0.9, 1.6, 2.3$ and 3. Figure 3.4 shows an image from the dataset in Table 3.8 distorted with blurring levels $B(1)$ to $B(5)$, where the distorted 640×480 images have PSNR 27.79, 27.18, 25.36, 24.14 and 23.49 respectively when compared with the original image. In practice, blur is typically introduced by changes of camera focus or by application of simple denoising techniques.

The speed-ups in execution time using SSD for this experiment are shown in Figure 3.7. FFT, FWHT and LRP_{sld} are the three fastest algorithms. FFT is the fastest for image size 160 × 120 with blurring levels $B(2)$ to $B(5)$ and for image sizes 320 × 240 and 640 × 480 with blurring levels $B(4)$ and $B(5)$, FWHT is the fastest for datasets $Scale4-1$ and $Scale5-2$ with blurring levels $B(1)$ to $B(2)$ and for dataset $Scale5-1$ with blurring levels $B(1)$ to $B(3)$, LRP_{sld} is the fastest in other cases. FWHT is faster than PGCK; PGCK is faster than PWHT. IDA is faster than PGCK for pattern size 16 × 16 and has similar speed-up as PGCK in other cases.

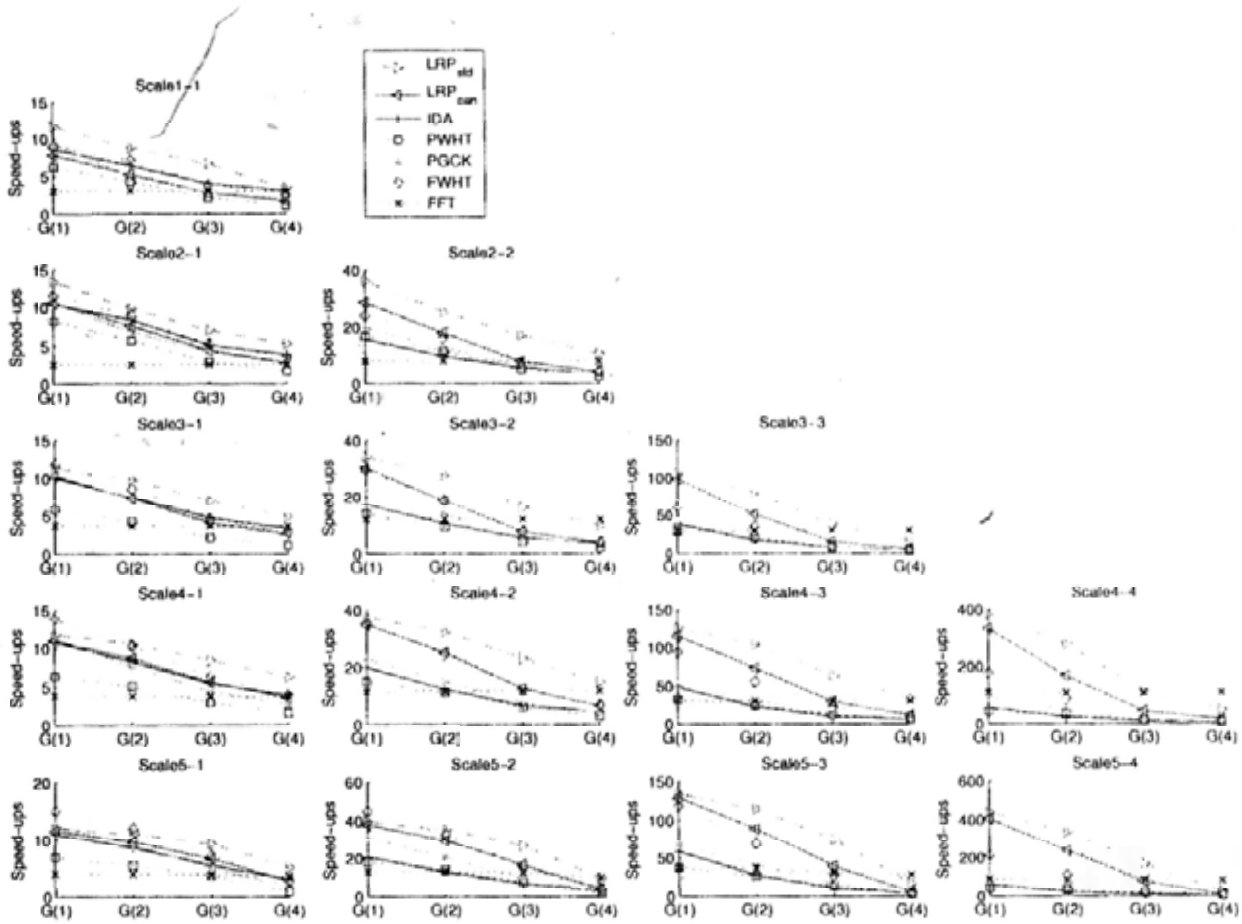


Figure 3.5: *Speed-ups in execution time for images with Gaussian noise when SSD is used.*

The speed-ups in number of operations are shown in Figure 3.8. It can be seen that the order of algorithm from the fastest to the slowest in number of operations is similar to the order in execution time. The main difference is that the algorithms, except FFT, have greater speed-up in number of operations than speed-up in execution time.

3.5.4 Experiment for JPEG Compressed Images

In this experiment, 5 different JPEG compression quality levels are used for each image of the datasets described in Table 3.8. The JPEG compression quality levels, which are referred to as $J(1)$, $J(2)$, $J(3)$, $J(4)$ and $J(5)$, correspond to quality measure $Q_{JPG} = 90, 70, 50, 30$ and 10 respectively , where higher Q_{JPG} means higher quality. Figure 3.4 shows an image from the dataset in Table 3.8 distorted with compression quality $J(1)$ to $J(5)$. $Q_{JPG} = 90, 70, 50, 30$ and 10 correspond to the compressed images having PSNR 39.88, 34.93, 33.10, 31.52 and 28.22 respectively when compared with the

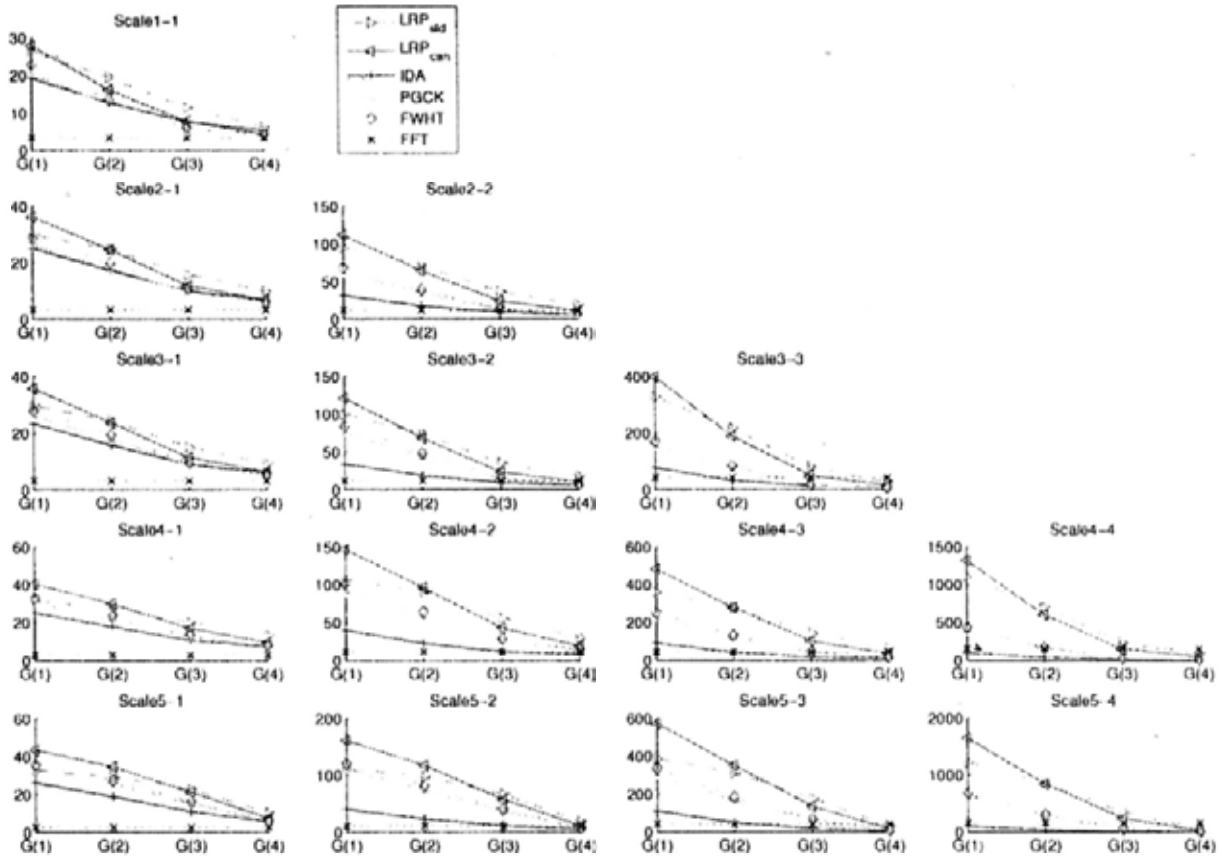


Figure 3.6: *Speed-ups in number of operations for images with Gaussian noise when SSD is used.*

original 640×480 image in Figure 3.4.

The speed-ups in execution time using SSD are shown in Figure 3.9. The performance of FFT algorithm is independent of compression quality while the speed-up of other fast algorithms decreases as the image quality decreases from $J(1)$ to $J(5)$. We can see that almost all the algorithms outperform the FFT algorithm with an exception that PWHT is outperformed by FFT in datasets $Scale4-4$ and $Scale5-4$. PGCK and FWHT are the fastest or close to the fastest in most cases for pattern size 16×16 and 32×32. LRP_{can} and FWHT are the fastest in most cases for pattern size 64×64 and 128×128.

The speed-ups in number of operations using SSD are shown in Figure 3.10. It can be seen the order of algorithm from the fastest to the slowest in number of operations is similar to the order in execution time. The main difference is that LRP_{can} in Figure 3.10 obviously requires fewer operations than other algorithms and the algorithms except

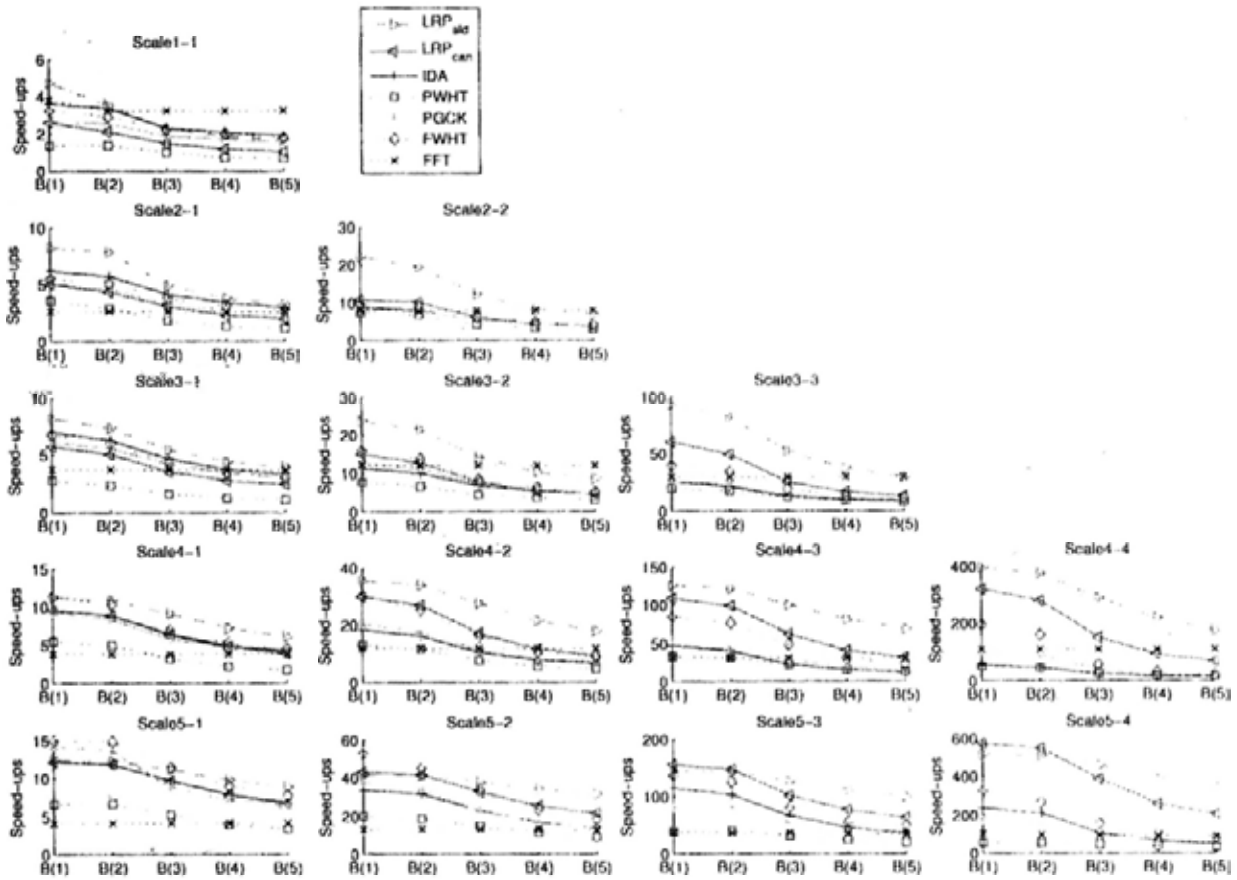


Figure 3.7: *Speed-ups in execution time for blurred images when SSD is used.*

FFT have greater speed-up in number of operations than speed-up in execution time.

3.5.5 Analysis of the Experimental Results when SSD is Used

The experimental results in Figure 3.2 - Figure 3.10 show some common properties of the evaluated FS-equivalent algorithms. 1) With pattern size fixed, the speed-ups over FS achieved by the fast algorithms increase by less than 2 as image size J increases by 4, e.g. from $Scale2-1$ to $Scale3-1$ and from $Scale3-1$ to $Scale4-1$. 2) With image size fixed, the speed-ups increase by about 2 to 4 as pattern size N increases by 4, e.g. from $Scale2-1$ to $Scale2-2$. 3) the speed-up for FFT is independent of distortions such as Gaussian noise, blur and JPEG compression. 4) The speed-up for IDA, PWHT, PGCK, FWHT and LRP decreases as the distortion level increases, e.g. from $G(1)$ to $G(4)$, because the difficulty in efficiently rejecting mismatching windows for these algorithms increases. 5) Except FFT, the ordering of the evaluated algorithms from the fastest

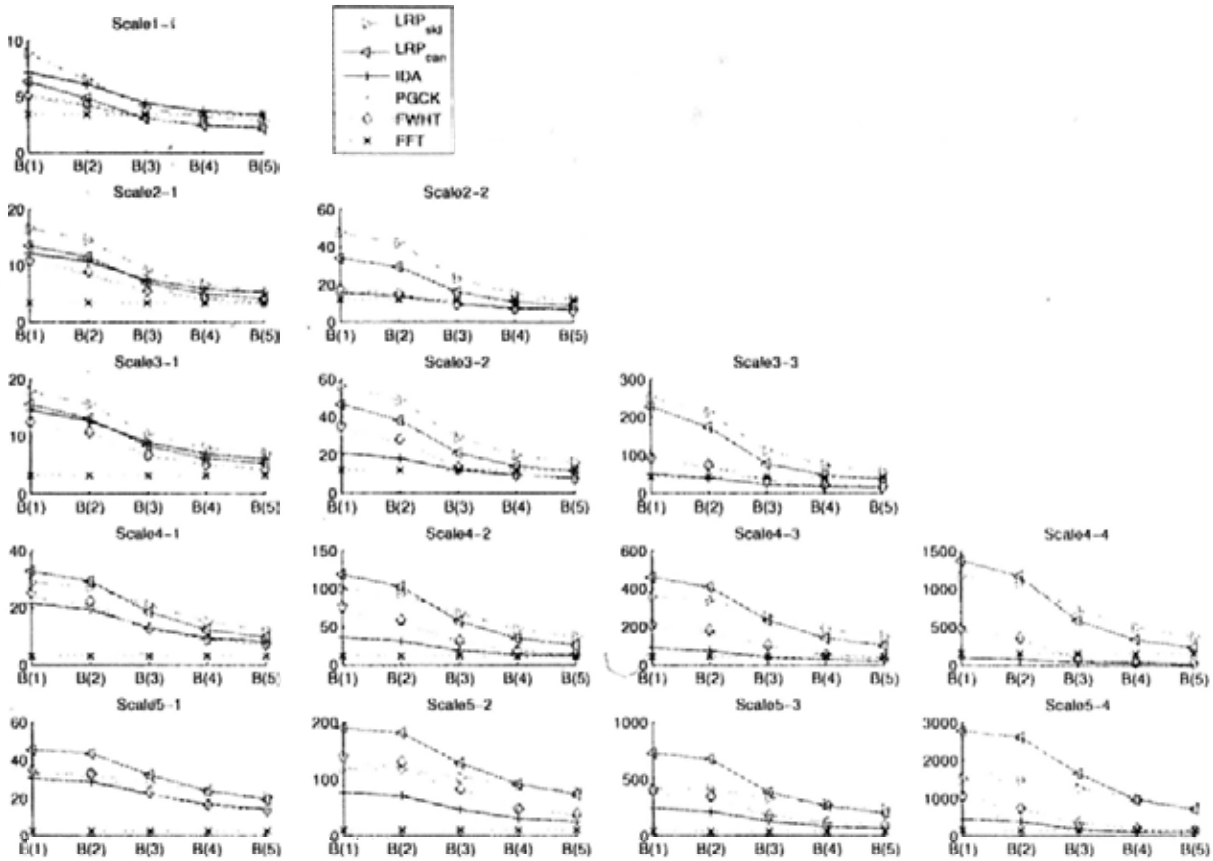


Figure 3.8: *Speed-ups in number of operations for blurred images when SSD is used.*

to the slowest in number of operations is similar to that in execution time. 6) The speed-up in execution time is close to the speed-up in number of operations for FFT while the speed-up in execution time is much smaller than the speed-up in number of operations for other algorithms, with the reason stated in the last paragraph of Section 3.5.2. 7) Despite the Y-axis difference, the comparative performances in number of operation and in execution time for PGCK, FWHT, IDA and LRP are similar for the same disturbance factor. The implementations of PGCK, FWHT, IDA and LRP have similar programming styles although they can be further optimized.

The fastest algorithm is different under different conditions. First, we consider using speed-up in execution time as the criteria. PGCK is the fastest for dataset without noise. FWHT is the fastest when pattern is small, image size is large and noise level is low, FFT is the fastest when noise is high, LRP_{std} is the fastest in other cases for datasets with Gaussian noise and blur. PGCK and FWHT are the fastest when

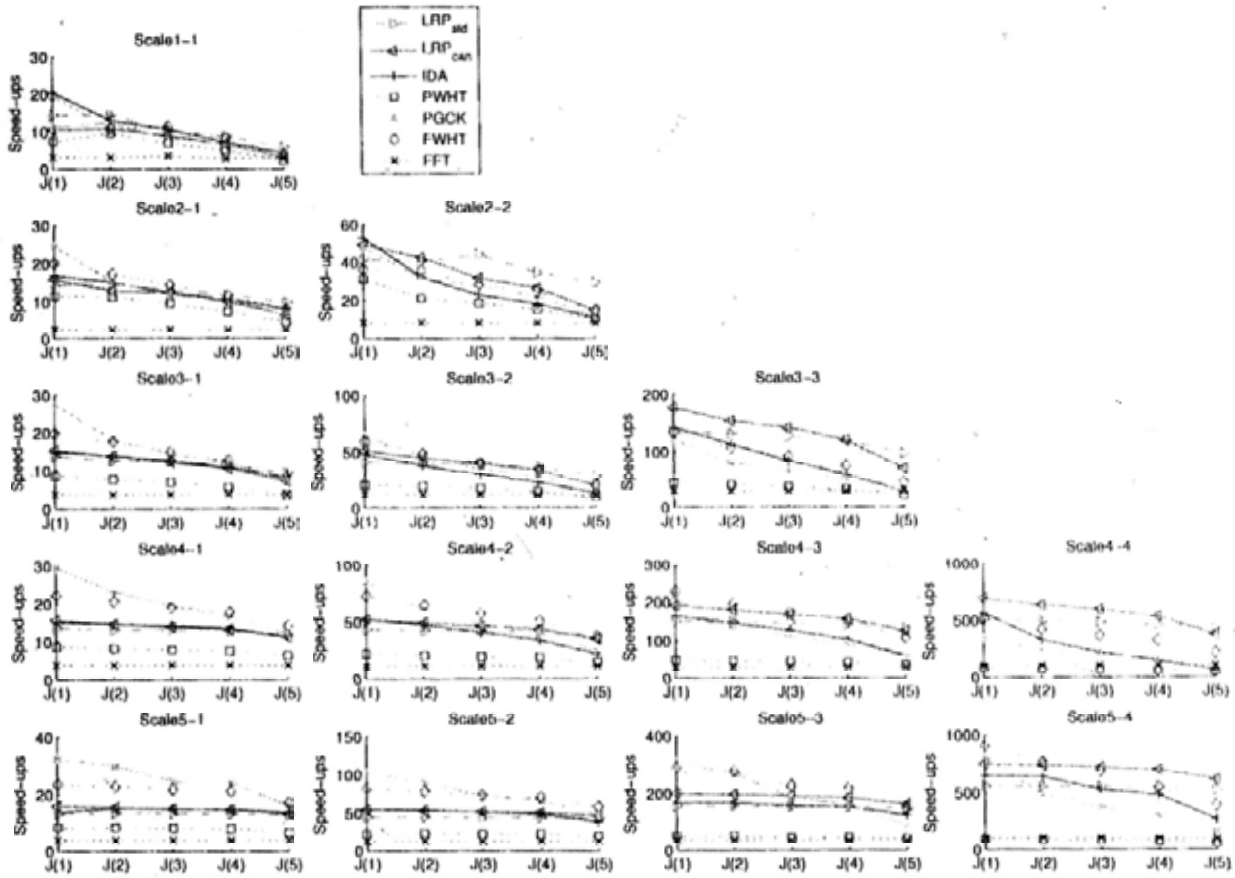


Figure 3.9: *Speed-ups in execution time for JPEG compressed images when SSD is used.*

pattern size is small or noise level is small while LRP_{can} is the fastest in other cases for datasets with JPEG compression. Second, we consider using speed-up in number of operations as the criteria. LRP_{sld} requires the least number of operations in most cases for datasets with Gaussian noise and blur. LRP_{can} requires the least number of operations for datasets with JPEG compression.

3.5.6 Analysis of the Experimental Results for Transform Domain Pattern Matching Algorithms

The higher is the distortion level, the more are basis vectors required for efficiently rejecting mismatching windows, and the more is the computation required by transform domain pattern matching algorithms in computing the transformation. Thus the speed-up for PWHT, PGCK, FWHT and LRP decreases as the distortion level increases. The dataset $Scale1-1$ with Gaussian noise in Figure 3.5 is taken as an example for

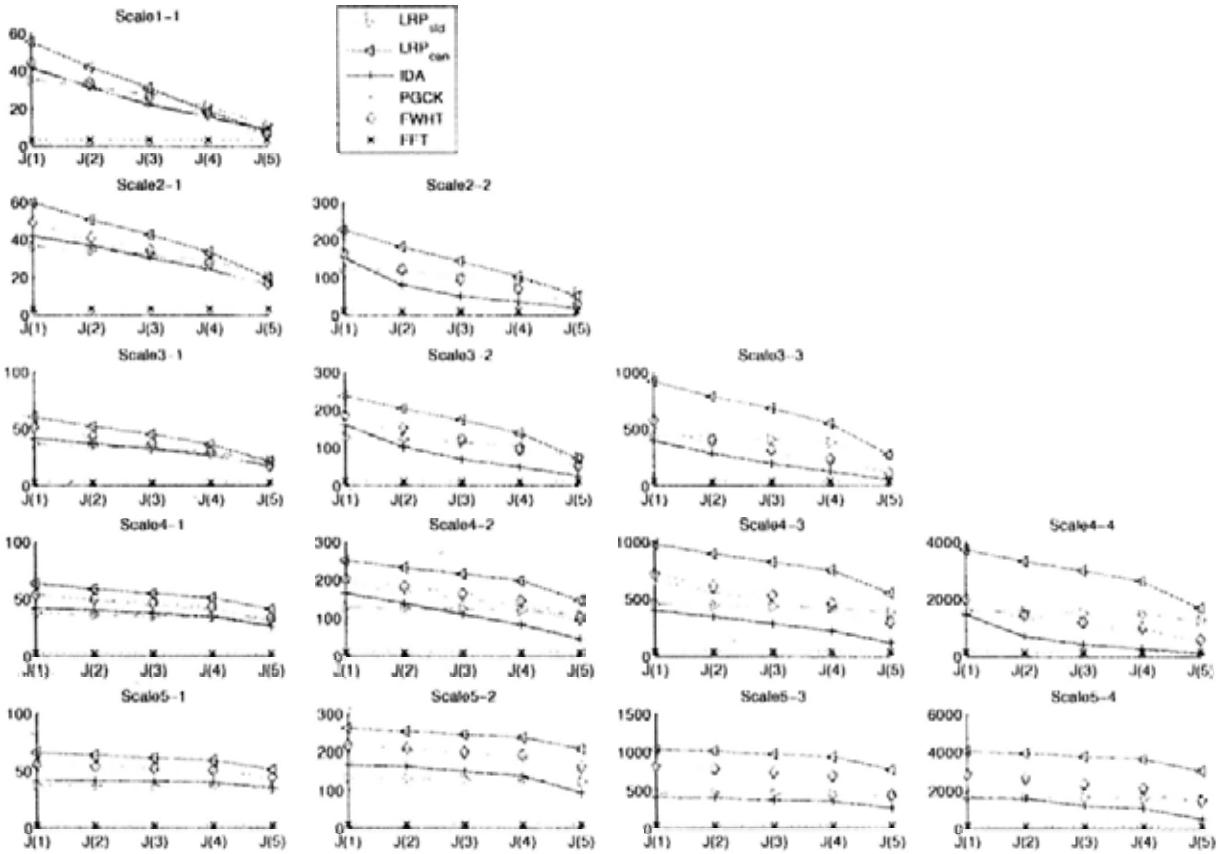


Figure 3.10: *Speed-ups in number of operations for JPEG compressed images when SSD is used.*

illustration. When the distortion level increases from $G(1)$ to $G(4)$, the average number of basis vectors required by PGCK increases from about 7 to about 23 and the execution time speed-up of PGCK over FS decreases from about 9 to about 2.5.

PGCK is the fastest for datasets without noise in Figure 3.2 and for datasets having low noise and small pattern size in Figure 3.9 because very few (less than 4) basis vectors are required for efficiently rejecting mismatching windows in these cases. In other cases, LRP and FWHT are faster than PGCK in computing transformation. Thus LRP and FWHT are faster than PGCK for most cases in Figure 3.5- Figure 3.10. As illustrated in Section 3.2.3, the more are basis vectors required for rejection, the more efficient is LRP compared with PWHT, PGCK and FWHT in computing the transformation. Thus LRP performs increasingly better compared with PWHT, PGCK and FWHT as the noise becomes larger from $G(1)$ to $G(4)$ in Figure 3.5 and 3.6, from $B(1)$ to $B(5)$ in Figure 3.7 and 3.8 or from $J(1)$ to $J(5)$ in Figure 3.9 and 3.10.

LRP_{sld} performs better than LRP_{can} as the noise becomes larger from $G(1)$ to $G(4)$ in Figure 3.5 and 3.6, from $B(1)$ to $B(5)$ in Figure 3.7 and 3.8 or from $J(1)$ to $J(5)$ in Figure 3.9 and 3.10. At loop k of the rejection step, LRP_{sld} and LRP_{can} require $3W$ and $3h^{(k-1)}N_{can}^{(k)}$ additions respectively for computing transformation. If the noise increases, then $N_{can}^{(k)}$ at loop k increases because the difficulty of rejection step in rejecting candidate windows increases. As $N_{can}^{(k)}$ increases, the $3h^{(k-1)}N_{can}^{(k)}$ additions required by LRP_{can} increase while the $3W$ additions required by LRP_{sld} keeps unchanged. Hence, LRP_{sld} is increasingly efficient than LRP_{can} in computing the transformation and in computing the pattern matching. Assume $h = 4$, $k = 2$ and a 16×16 pattern is searched in a 256×256 image, we have $N = 256$, $J = 65536$ and $N_{can}^{(1)} = W = (256 - 16 + 1)^2 = 58081$. Assume $N_{can}^{(2)} = 10$ in the first example, then LRP_{sld} requires $3W = 174243$ additions and LRP_{can} requires $3h^{(k-1)}N_{can}^{(k)} = 3 \times 4^{(2-1)} \times 10 = 120$ additions in computing transformation at loop $k = 2$. Thus LRP_{can} is more efficient than LRP_{sld} in computing transformation for this example. In this situation, the rejection step is so efficient that $N_{can}^{(1)} - N_{can}^{(2)} = 58071$ mismatched candidates are eliminated at the first loop of k. Situations similar to this example usually happen when noise level is small. In the second example, assume $N_{can}^{(2)}$ increases from 10 in the first example to $11W/12$ in this example, then LRP_{sld} requires $3W$ additions and LRP_{can} requires $3 \times 4^{(2-1)} \times 11W/12 = 11W$ additions for computing transformation in loop $k = 2$. Thus LRP_{sld} is more efficient than LRP_{can} in computing transformation for this example. In this situation, the rejection step is so inefficient that only $N_{can}^{(1)} - N_{can}^{(2)} = W/12$ candidates are eliminated at the first loop of k. Situations similar to this example usually happen when larger noises are added.

Compared with other transform domain pattern matching algorithms, LRP_{can} performs relatively worse in execution time than in number of operation. The main reason for this phenomenon is that LRP_{can} jumpily computes the transformation for the remaining candidates, which results in higher miss penalties of cache memories than the other approaches, e.g. LRP_{sld}, that compute the transformation in sliding window manner.

3.5.7 Experimental Results when SAD is Used

For the experiments in this section, speed-ups in execution time are evaluated using SAD as the dissimilarity measure. The perturbations correspond to those introduced in Section 3.5.2 - Section 3.5.4. Since the FFT approach cannot be applied when SAD is used, it does not appear in the figures.

Figure 3.11 shows the results for images with Gaussian noise. IDA is the fastest for pattern size 16×16 while LRP_{can} and LRP_{sld} are the fastest and have similar performance in other cases.

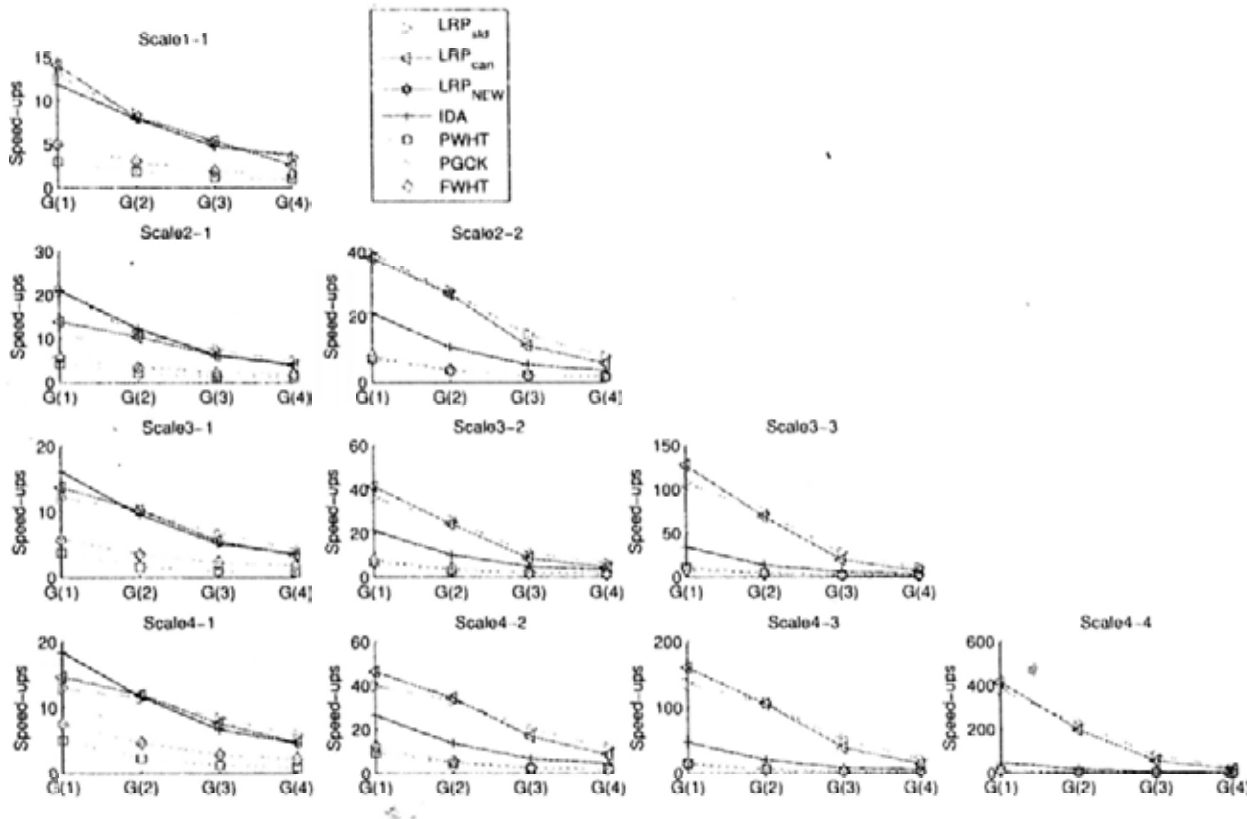


Figure 3.11: *Speed-ups in execution time for images with Gaussian noise when SAD is used.*

The experimental results for low pass filter blurred images in Figure 3.12 show that the algorithms from the fastest to the slowest for pattern size 16×16 can be ordered as: 1) IDA, 2) LRP_{sld}, 3) LRP_{can}, 4) PGCK and FWHT, 5) PWHT. The order for other pattern sizes is: 1) LRP_{sld} or LRP_{can}, 2) IDA, 3) PWHT, PGCK and FWHT, where LRP_{can} is the fastest for datasets $Scale3-3$, $Scale4-2$, $Scale4-3$ and $Scale4-4$ with noise levels $B(1)$ and $B(2)$ while LRP_{sld} is the fastest for other cases.

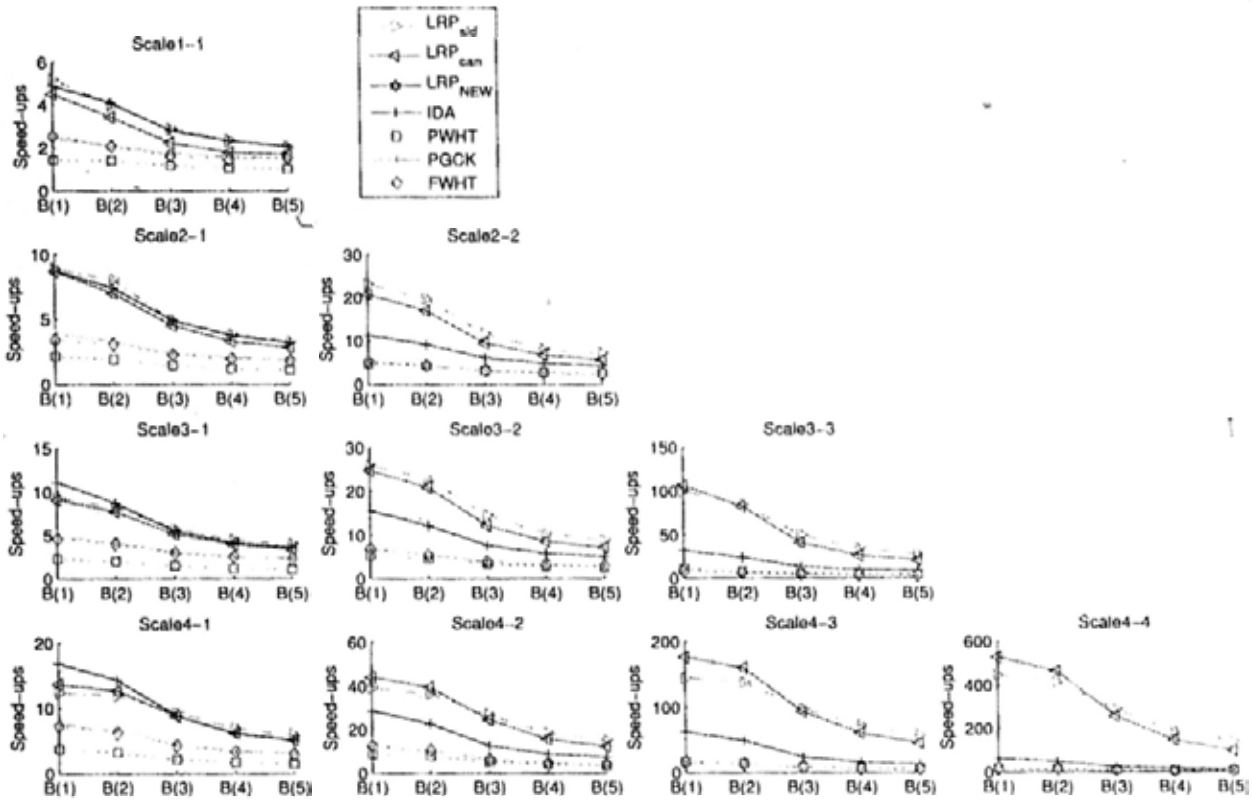


Figure 3.12: *Speed-ups in execution time for blurred images when SAD is used.*

The experimental results for JPEG compressed images in Figure 3.12 show that IDA is the fastest for pattern size 16×16. IDA is also the fastest for quality levels $J(1)$ and $J(2)$ in all datasets except that it is outperformed by LRP_{can} for $J(2)$ in $Scale3-3$ and $Scale4-4$. Otherwise, when pattern size is larger than 16×16 and noise level is high, LRP_{can} is the fastest in most cases.

In summary, IDA is the fastest in execution time when pattern size is 16×16 while LRP_{sld} is the fastest in larger pattern sizes for dataset with Gaussian noise and Gaussian low pass filter. LRP_{can} is the fastest for large pattern size and high noise level while IDA is the fastest in other cases for JPEG compressed images. The comparative performance of LRP_{sld}, LRP_{can} and IDA using SAD is similar to that using SSD for the three noise types. PWHT, PGCK and FWHT are inefficient in the experiments when SAD is used and performs better when SSD is used.

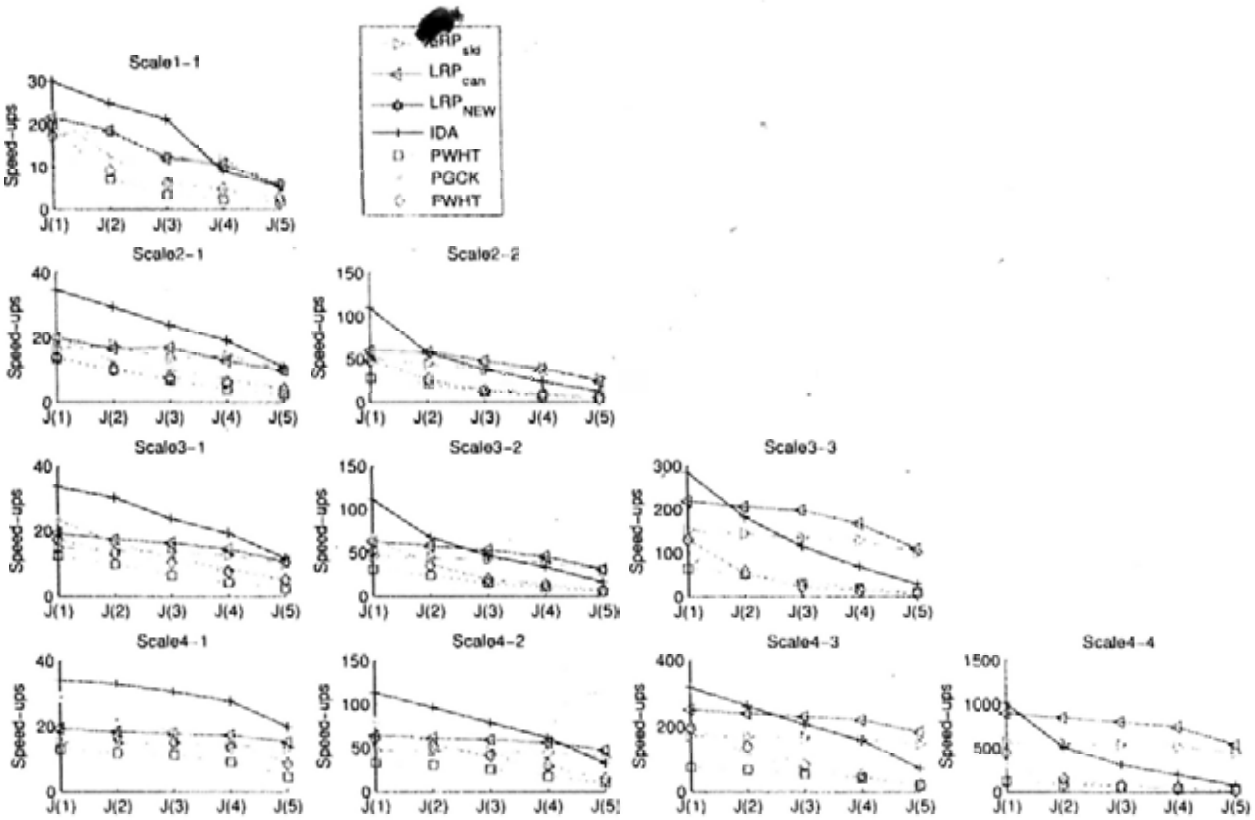


Figure 3.13: *Speed-ups in execution time for JPEG compressed images when SAD is used.*

3.5.8 Termination Strategy Comparison

In this experiment, we compare the proposed termination strategy with the strategy in [1]. PGCK and FWHT are used as the testing algorithms. SSD is used as the dissimilarity measure. The images with JPEG compression quality levels $J(1)$ to $J(5)$ as introduced in Section 3.5.4 for the datasets in Table 3.8 are used as the testing datasets. The speed-ups are average of the speed-ups in the environments introduced in Table 3.9. The experimental results in Figure 3.14 show that PGCK/FWHT using the proposed termination strategy is faster than PGCK/FWHT using the strategy in [1] when pattern size is larger than 16×16.

3.6 Discussions

3.6.1 Summary of the evaluation results

First, we consider execution time as the criterion when SSD is used as the dissimilarity measure. PGCK is the fastest for dataset without noise. FWHT is the fastest when

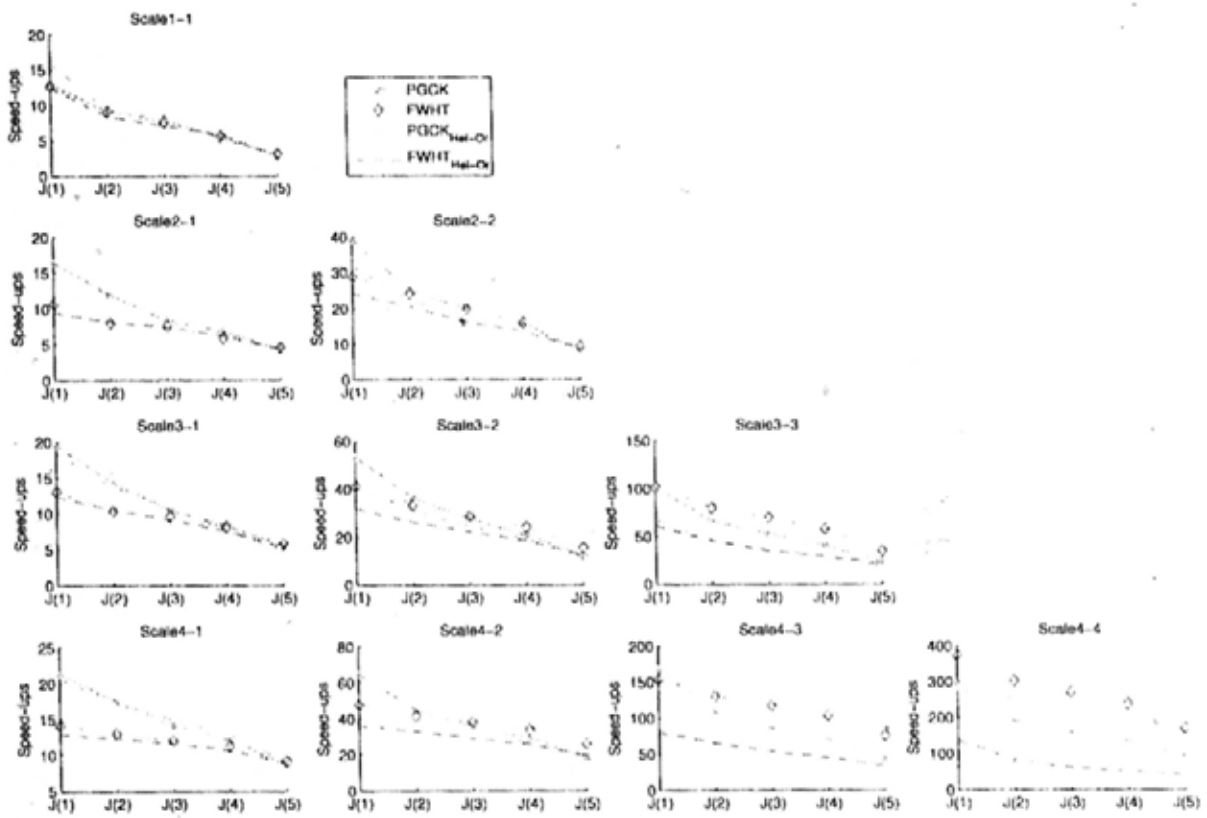


Figure 3.14: *Comparison of termination strategies for JPEG compressed images using SSD.* ***PGCK and FWHT****: results using the proposed strategy;* $\mathbf{PGCK}_{Hel-Or}$ **and** $\mathbf{FWHT}_{Hel-Or}$*: results using Hel-Or and Hel-Or's strategy in [1].*

pattern is small, image size is large and noise level is low, FFT is the fastest when noise is high and LRP_{sld} is the fastest in other cases for datasets with Gaussian noise and blur. PGCK and FWHT are the fastest when pattern size is small or noise level is small while LRP_{can} is the fastest in other cases for datasets with JPEG compression.

Second, we consider execution time as the criterion when SAD is used as the dissimilarity measure. IDA is the fastest in execution time when pattern size is 16×16 while LRP_{sld} is the fastest in larger pattern sizes for dataset with Gaussian noise and Gaussian low pass filter. LRP_{can} is the fastest for large pattern size and high noise level while IDA is the fastest in other cases for JPEG compressed images.

Table 3.10 summarizes the aforementioned results.

Finally, we consider computational complexity based on number of operations as the criterion. The tests performed with SSD reported that LRP_{sld} requires the smallest number of operations for dataset distorted by Gaussian noise and blur, while LRP_{can}

	No Disturbance	Gaussian Noise	Blur	JPEG compression
SSD	PGCK	LRP / FWHT / FFT	LRP / FWHT / FFT	LRP / PGCK / FWHT
SAD	PGCK / IDA	LRP / IDA	LRP / IDA	LRP / IDA

Table 3.10: *Best overall algorithms as for measured execution times for different disturbance factors and matching measures.*

requires the smallest number of operations for dataset with JPEG compression.

3.6.2 Miscellaneous properties of the evaluated algorithms

Although this chapter mainly focuses on the computational efficiency of the compared algorithms, we list the miscellaneous properties of these algorithms so that suitable choice of algorithm can be made for specific applications:

1. PWHT, PGCK and FWHT can help deal with illumination effect while FS, IDA, FFT and LRP cannot. This property of PWHT is used in wide baseline image matching [15].

2. LRP, PWHT, PGCK and FWHT can help deal with multi-scale pattern matching when the size of candidate window are the integer multiples of the size of the pattern or vise versa while FS, IDA and FFT cannot.

3. Sometimes, the pattern to be matched may not be rectangular, the method proposed by Ben-Yehuda *et al.* in [35] helps PWHT, PGCK and FWHT to deal with this problem by segmenting the pattern into multiple dyadic components. The irregularity of pattern will have no influence on FS, will have small influence on IDA and will have much influence on algorithms FFT, LRP, PWHT, PGCK and FWHT that require input data size to be $a \cdot 2^b$ for $a = 1, 2, \ldots, b = 1, 2, \ldots$.

3.7 Summary

In this chapter, we have presented execution time evaluation and computational complexity analysis of recent FS-equivalent algorithms. The algorithms have been compared considering different sizes of images and patterns in presence of Gaussian noise, image blur and JPEG compression. Our experimental results show clearly that the fastest algorithm is different under different conditions. Nonetheless, experimental evidence suggests also that, overall, LRP may be considered the best performing algorithm, for

it turns out to be the fastest in most cases. This nicely agrees with the theoretical analysis developed in Theorem 3.1, which proves why LRP is faster than other recent approaches that use WHT for transform domain pattern matching in most cases. Throughout this chapter, we have discussed the motivations underpinning the relative merits and limits of the considered algorithms under the different working conditions, so as to possibly inspire the development of new methods in the active research field of fast full search equivalent pattern matching. Inspired by this analysis, we proposed new algorithms in Chaper 4 and Chapter 5.

3.8 Appendix A: Proof of Theorem 3.1

Theorem 3.1 When the $u = 2^n$ basis vectors in $\mathbf{V}_{WHT}^{(u\times N)}$ are the first sequency-ordered WHT basis vectors, we have the LRP transform matrix:

$$\mathbf{V}_{LRP}^{(u\times N)} = \mathbf{I}_u \otimes \mathbf{1}_{1\times(N/u)}, \tag{3.20}$$

such that: 1) the subspace spanned by the u basis vectors in $\mathbf{V}_{WHT}^{(u\times N)}$ is equal to the subspace spanned by the u basis vectors in $\mathbf{V}_{LRP}^{(u\times N)}$; 2) for any length-$N$ input vector $\vec{\mathbf{x}}$, if the basis vectors in $\mathbf{V}_{LRP}^{(u\times N)}$ and $\mathbf{V}_{WHT}^{(u\times N)}$ are normalized to have L_2 norm equal to 1, then $||\mathbf{V}_{WHT}^{(u\times N)}\vec{\mathbf{x}}||_2^2 = ||\mathbf{V}_{LRP}^{(u\times N)}\vec{\mathbf{x}}||_2^2$; 3) the transformation $\mathbf{V}_{WHT}^{(u\times N)}\vec{\mathbf{x}}$ requires at least $3u/2$ additions per pixel while the transformation $\mathbf{V}_{LRP}^{(u\times N)}\vec{\mathbf{x}}$ requires 3 additions per pixel when computed on sliding windows.

Proof: The following equation for sequency ordered WHT is proved in [33]:

$$\vec{\mathbf{m}}^{(N,ai+b)} = \vec{\mathbf{m}}^{(a,f(b,a,i))} \otimes \vec{\mathbf{m}}^{(N/a,i)},$$
$$\text{where } f(b,a,i) = \begin{cases} b, & \text{i is an even number,} \\ a-1-b, & \text{i is an odd number.} \end{cases} \tag{3.21}$$

When $a = u, i = 0$ in (3.21), we have:

$$\vec{\mathbf{m}}^{(N,b)} = \vec{\mathbf{m}}^{(N,u\cdot 0+b)} = \vec{\mathbf{m}}^{(u,b)} \otimes \vec{\mathbf{m}}^{(N/u,0)}. \tag{3.22}$$

Since $\vec{\mathbf{m}}^{(N/u,0)} = \mathbf{1}_{1\times(N/u)}$, we have the follows from (3.22):

$$\begin{aligned}\mathbf{V}_{WHT}^{(u\times N)} &= \mathbf{M}^{(u)} \otimes \vec{\mathbf{m}}^{(N/u,0)} = (\mathbf{M}^{(u)} \otimes \mathbf{1}_{1\times(N/u)}) \\ &= (\mathbf{M}^{(u)}\mathbf{I}_u) \otimes (\mathbf{I}_1\mathbf{1}_{1\times(N/u)}) \\ &= (\mathbf{M}^{(u)} \otimes \mathbf{I}_1)(\mathbf{I}_u \otimes \mathbf{1}_{1\times(N/u)}) \\ &= \mathbf{M}^{(u)}(\mathbf{I}_u \otimes \mathbf{1}_{1\times(N/u)}).\end{aligned} \tag{3.23}$$

Comparing WHT matrix in (3.23) and LRP matrix in (3.20), we have:

$$\mathbf{V}_{WHT}^{(u\times N)} = \mathbf{M}^{(u)}\mathbf{V}_{LRP}^{(u\times N)}. \tag{3.24}$$

According to (3.24), the orthogonal basis vectors in $\mathbf{V}_{WHT}^{(u\times N)}$ can be represented by orthogonal basis vectors in $\mathbf{V}_{LRP}^{(u\times N)}$ and vise versa, so the subspace spanned by the u basis vecotrs in $\mathbf{V}_{WHT}^{(u\times N)}$ is equal to the subspace spanned by the u basis vectors in $\mathbf{V}_{LRP}^{(u\times N)}$.

We can normalize the orthogonal basis vectors in $\mathbf{V}_{LRP}^{(u\times N)}$ and have orthonormal basis vectors in $\frac{\mathbf{V}_{LRP}^{(u\times N)}}{||\mathbf{V}_{LRP}^{(u\times N)}||_2}$. We can normalize the orthogonal basis vectors in $\mathbf{V}_{WHT}^{(u\times N)}$ and have orthonormal basis vectors in $\frac{\mathbf{V}_{WHT}^{(u\times N)}}{||\mathbf{V}_{WHT}^{(u\times N)}||_2}$. The orthonormal basis vectors in $\frac{\mathbf{V}_{WHT}^{(u\times N)}}{||\mathbf{V}_{WHT}^{(u\times N)}||_2}$ and the orthonormal basis vectors in $\frac{\mathbf{V}_{LRP}^{(u\times N)}}{||\mathbf{V}_{LRP}^{(u\times N)}||_2}$ are spanning the same subspace, so we have the follows for any input vector $\vec{\mathbf{x}}$:

$$||\frac{\mathbf{V}_{WHT}^{(u\times N)}}{||\mathbf{V}_{WHT}^{(u\times N)}||_2}\vec{\mathbf{x}}||_2^2 = ||\frac{\mathbf{V}_{LRP}^{(u\times N)}}{||\mathbf{V}_{LRP}^{(u\times N)}||_2}\vec{\mathbf{x}}||_2^2, \tag{3.25}$$

where $||\mathbf{V}_{LRP}^{(u\times N)}||_2^2 = \frac{N}{u}$ and $||\mathbf{V}_{WHT}^{(u\times N)}||_2 = N$. Thus when orthogonal basis vectors are normalized to be orthonormal, the energy extracted from $\vec{\mathbf{x}}$ using the basis vectors in $\mathbf{V}_{LRP}^{(u\times N)}$ is the same as that using the basis vectors in $\mathbf{V}_{WHT}^{(u\times N)}$.

Consider the computational complexity, at least $3u/2$ additions per pixel are required for the transformation $\mathbf{V}_{WHT}^{(u\times N)}\vec{\mathbf{x}}$ using the FWHT algorithm in [33]; the transformation $\mathbf{V}_{LRP}^{(u\times N)}\vec{\mathbf{x}}$ is actually the sum of pixel values in rectangles on sliding windows which can be computed by 3 additions per pixel using the integral image method [48].

Chapter 4

The Kronecker-Hadamard Transform for Fast Pattern Matching

4.1 Introduction

The Gray-Code Kernels (GCK) family which has Walsh Hadamard Transform (WHT) on sliding windows as a member is a family of kernels that can perform image analysis efficiently using a fast algorithm such as the GCK algorithm [32]. The GCK has been successfully used for pattern matching. In this chapter, we develop a new family of transforms called Kronecker-Hadamard Transform (KHT) of which the GCK and WHT are special cases. A fast KHT algorithm is also developed for KHT, which requires 4/3 additions per datum per basis vector independent of transform size and dimension. All KHT, like GCK and WHT, can be computed efficiently using the fast KHT algorithm. The fast KHT algorithm assumes that one projection value on all window positions have been computed by other approaches. This assumption is the same as that of the GCK algorithm. Limited by WHT or GCK, fast algorithms such as the GCK algorithm, our previous algorithm in [33] and the KHT algorithm require more than one addition to compute one basis vector. We find that a subset of KHT, which will be called Segmented KHT (SegKHT), can be computed using a fast algorithm that is more efficient than the fast KHT algorithm. By segmenting input data into L_s parts, the fast SegKHT algorithm requires $4/(3L_s)$ addition(s) per datum per basis vector. For example, when L_s is 8, the SegKHT requires only 1/6 addition per datum per basis vector. SegKHT is a subset of KHT that cannot be represented by GCK. The KHT and SegKHT because of their fast algorithms can be applied in real time applications such as pattern matching, object detection, feature extraction, texture analysis/synthesis.

The rest of the chapter is organized as follows. Section 4.2 introduces KHT. Section 4.3 illustrates the fast KHT algorithm. Section 4.4 introduces the SegKHT and its fast

algorithm. Section 4.5 analyzes the advantage of KHT over WHT and GCK using two examples. Section 4.6 gives the experimental results. Finally, Section 4.7 presents the summary.

4.2 The Kronecker-Hadamard Transform

4.2.1 The WHT

1D order-N_M WHT transforms N_M numbers into N_M projection values. Let $\mathbf{M}^{(N_M)}$ be an order-N_M WHT matrix and:

$$\begin{aligned}\mathbf{M}^{(N_M)} &= [\vec{\mathbf{m}}^{(N_M,0)}, \ldots \vec{\mathbf{m}}^{(N_M,i_M)}, \ldots \vec{\mathbf{m}}^{(N_M,N_M-1)}]^T \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} \mathbf{M}^{(N_M/2)} & \mathbf{M}^{(N_M/2)} \\ \mathbf{M}^{(N_M/2)} & -\mathbf{M}^{(N_M/2)} \end{bmatrix},\end{aligned} \tag{4.1}$$

where $\mathbf{M}^{(1)} = 1$, $\mathbf{M}^{(N_M)}$ is an $N_M \times N_M$ matrix, $\vec{\mathbf{m}}^{(N_M,i_M)}$ for $i_M = 0, \ldots, N_M - 1$ is the i_Mth WHT basis vector having length N_M. $\vec{\mathbf{m}}^{(N_M,i_M)^T}$ is the i_Mth row of $\mathbf{M}^{(N_M)}$ in (4.1). When $N_M = 8$, Fig. 4.1 shows the order-8 WHT in dyadic order and sequency order. The dyadic order and the sequency order are different methods for ordering WHT basis vectors [58]. For example, the $\vec{\mathbf{m}}^{(N_M,2)}$ for dyadic-ordered WHT is the $\vec{\mathbf{m}}^{(N_M,3)}$ for sequency-ordered WHT. For sequency-ordered WHT, the extracted spatial frequency increases as the index i_M of basis vector $\vec{\mathbf{m}}^{(N_M,i_M)}$ increases. The relationship between dyadic-ordered and sequency-ordered WHT is detailed in [52]. For ease of explanation, dyadic-ordered WHT will be used in the followings of this chapter if not specified.

WHT has long been used for image representation under numerous applications. More discussions on applying WHT for pattern matching are available in [1; 51; 59].

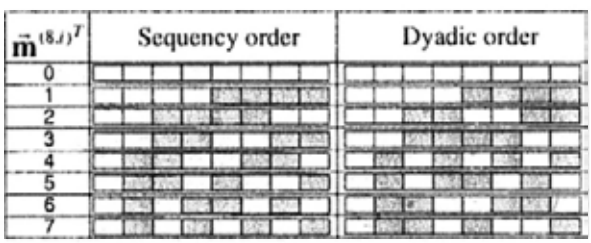


Figure 4.1: *Order-8 WHT basis vectors in sequency order and dyadic order. White represents the value +1 and grey represents the value −1. Normalization factor of basis vectors is skipped.*

4.2.2 Definition of 1D KHT

The Kronecker-Hadamard Transform (KHT) is best explained using Kronecker product which is denoted as $\otimes$. If $\mathbf{A}$ is a $U_1 \times Q_1$ matrix (a_{n_1,n_2}) and $\mathbf{B}$ is a $U_2 \times Q_2$ matrix (b_{m_1,m_2}), then $\mathbf{A} \otimes \mathbf{B}$ is a $U_1U_2 \times Q_1Q_2$ matrix as follows:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{0,0}\mathbf{B} & a_{0,1}\mathbf{B} & \cdots & a_{0,Q_1-1}\mathbf{B} \\ a_{1,0}\mathbf{B} & a_{1,1}\mathbf{B} & \cdots & a_{1,Q_1-1}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{U_1-1,0}\mathbf{B} & a_{U_1-1,1}\mathbf{B} & \cdots & a_{U_1-1,Q_1-1}\mathbf{B} \end{bmatrix}. \tag{4.2}$$

Kronecker product has been successfully used for designing algorithms such as Strassen's matrix multiplication [60] and fast DCT algorithms [61]. More information on Kronecker product can be found in [62].

Let matrices $\mathbf{S}_l \in \mathbb{R}^{L \times L}$ and $\mathbf{S}_r \in \mathbb{R}^{R \times R}$ be:

$$\begin{aligned} \mathbf{S}_l &= [\vec{\mathbf{s}}_l^{(0)}, \ldots, \vec{\mathbf{s}}_l^{(i_l)}, \ldots, \vec{\mathbf{s}}_l^{(L-1)}]^T, \\ \mathbf{S}_r &= [\vec{\mathbf{s}}_r^{(0)}, \ldots, \vec{\mathbf{s}}_r^{(i_r)}, \ldots, \vec{\mathbf{s}}_r^{(R-1)}]^T. \end{aligned} \tag{4.3}$$

Define the 1D order-N KHT matrix of $\mathbf{S}_l$, WHT matrix $\mathbf{M}^{(N_\mathbf{M})}$ and $\mathbf{S}_r$ as:

$$\begin{aligned} \mathbf{V}^{(N)} &= [\vec{\mathbf{v}}^{(N,0)}, \ldots \vec{\mathbf{v}}^{(N,i)}, \ldots \vec{\mathbf{v}}^{(N,N-1)}]^T \\ &= \mathbf{S}_l \otimes \mathbf{M}^{(N_\mathbf{M})} \otimes \mathbf{S}_r, \qquad (4.4) \\ \text{where} \quad & N = LN_\mathbf{M}R, \qquad (4.5) \end{aligned}$$

subscript $\cdot_l$ denotes the left matrix, $\cdot_r$ denotes the right matrix and $\vec{\mathbf{v}}^{(N,i)}$ for $i = 0, \ldots, N-1$ denotes the ith KHT basis vector given by $\vec{\mathbf{s}}_l^{(i_l)}$, $\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M})}$ and $\vec{\mathbf{s}}_r^{(i_r)}$ as follows:

$$\vec{\mathbf{v}}^{(N,i)} = \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M})} \otimes \vec{\mathbf{s}}_r^{(i_r)}. \tag{4.6}$$

If matrices $\mathbf{A_1}$, $\mathbf{A_2}$ and $\mathbf{A_3}$ have sizes $U_1 \times Q_1$, $U_2 \times Q_2$ and $U_3 \times Q_3$ respectively, then $\mathbf{A_1} \otimes \mathbf{A_2} \otimes \mathbf{A_3}$ is an $U_1U_2U_3 \times Q_1Q_2Q_3$ matrix. $\mathbf{S}_l$, $\mathbf{M}^{(N_\mathbf{M})}$ and $\mathbf{S}_r$ have sizes $L \times L$, $N_\mathbf{M} \times N_\mathbf{M}$ and $R \times R$ respectively, so the KHT matrix $\mathbf{V}^{(N)}$ is an $N \times N$ matrix, where $N = LN_\mathbf{M}R$. For example, let $\mathbf{S}_l = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $\mathbf{M}^{(N_\mathbf{M})} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ and $\mathbf{S}_r = [1]$.

Hence,

$$\begin{aligned}\mathbf{V}^{(4)} &= \mathbf{S}_l \otimes \mathbf{M}^{(2)} \otimes \mathbf{S}_r \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & -1 & 2 & -2 \\ 3 & 3 & 4 & 4 \\ 3 & -3 & 4 & -4 \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{v}}^{(4,0)^T} \\ \vec{\mathbf{v}}^{(4,1)^T} \\ \vec{\mathbf{v}}^{(4,2)^T} \\ \vec{\mathbf{v}}^{(4,3)^T} \end{bmatrix}. \end{aligned} \tag{4.7}$$

Consider J input elements x_j for $j = 0, 1, \ldots, J-1$, which are divided into overlapping windows of size N ($J > N$). Let the jth length-N input window for $j = 0, 1, \ldots, J-N$ be:

$$\vec{\mathbf{x}}_w^{(N,j)} = [x_j, x_{j+1}, \ldots, x_{j+N-1}]^T. \tag{4.8}$$

For 1D pattern matching , $\vec{\mathbf{x}}_w^{(N,j)}$ are sliding candidate windows which will be compared with the given pattern $\vec{\mathbf{x}}_t^{(N)}$. Let $y(N,i,j)$ for $i = 0, 1, \ldots, N-1; j = 0, 1, \ldots, J-N$ be the ith KHT projection value for the jth window and

$$y(N,i,j) = < \vec{\mathbf{v}}^{(N,i)}, \vec{\mathbf{x}}_w^{(N,j)} > = \vec{\mathbf{v}}^{(N,i)^T} \vec{\mathbf{x}}_w^{(N,j)}. \tag{4.9}$$

In other words, $y(N,i,j)$ is the projection of the jth input window $\vec{\mathbf{x}}_w^{(N,j)}$ defined in (4.8) onto the ith KHT basis vector $\vec{\mathbf{v}}^{(N,i)}$ defined in (4.6). Let $\vec{\mathbf{y}}^{(N,j)}$ be the projection value vector containing all order-N KHT projection values at the jth window and

$$\begin{aligned}\vec{\mathbf{y}}^{(N,j)} &= \mathbf{V}^{(N)} \vec{\mathbf{x}}_w^{(N,j)} \\ &= \begin{bmatrix} y(N,0,j) \\ y(N,1,j) \\ \vdots \\ y(N,N-1,j) \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{v}}^{(N,0)^T} \\ \vec{\mathbf{v}}^{(N,1)^T} \\ \vdots \\ \vec{\mathbf{v}}^{(N,N-1)^T} \end{bmatrix} \vec{\mathbf{x}}_w^{(N,j)}. \end{aligned} \tag{4.10}$$

For example, the projection value vector for the $\mathbf{V}^{(N)}$ in (4.7) is:

$$\begin{aligned}\vec{\mathbf{y}}^{(4,j)} &= \mathbf{V}^{(4)}\vec{\mathbf{x}}_w^{(4,j)} \\ &= \begin{bmatrix} y(4,0,j) \\ y(4,1,j) \\ y(4,2,j) \\ y(4,3,j) \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & -1 & 2 & -2 \\ 3 & 3 & 4 & 4 \\ 3 & -3 & 4 & -4 \end{bmatrix} \begin{bmatrix} x_j \\ x_{j+1} \\ x_{j+2} \\ x_{j+3} \end{bmatrix}.\end{aligned} \tag{4.11}$$

4.2.3 Properties of KHT

Denote the $N \times N$ identity matrix by $\mathbf{I}_N$. The following property shows how to construct the N orthonormal basis vectors in the $N \times N$ KHT matrix for the transform domain pattern matching introduced in Section 1.2.4.

Property 4.1 For the KHT matrix $\mathbf{V}^{(N)} = \mathbf{S}_l \otimes \mathbf{M}^{(N_\mathbf{M})} \otimes \mathbf{S}_r$ defined in (4.4), if both $\mathbf{S}_l$ and $\mathbf{S}_r$ are orthogonal, i.e. $\mathbf{S}_l\mathbf{S}_l^T = \mathbf{I}_L$ and $\mathbf{S}_r\mathbf{S}_r^T = \mathbf{I}_R$, then the KHT matrix $\mathbf{V}^{(N)}$ is orthogonal.

Proof: The following properties of Kronecker product are used for this proof:

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC}) \otimes (\mathbf{BD}), \tag{4.12}$$

$$(\mathbf{A} \otimes \mathbf{B})^T = \mathbf{A}^T \otimes \mathbf{B}^T. \tag{4.13}$$

If $\mathbf{S}_l\mathbf{S}_l^T = \mathbf{I}_L$ and $\mathbf{S}_r\mathbf{S}_r^T = \mathbf{I}_R$, then we have:

$$\begin{aligned}&\mathbf{V}^{(N)}\mathbf{V}^{(N)T} \\ &= (\mathbf{S}_l \otimes \mathbf{M}^{(N_\mathbf{M})} \otimes \mathbf{S}_r)(\mathbf{S}_l \otimes \mathbf{M}^{(N_\mathbf{M})} \otimes \mathbf{S}_r)^T \\ &= (\mathbf{S}_l \otimes \mathbf{M}^{(N_\mathbf{M})} \otimes \mathbf{S}_r)(\mathbf{S}_l^T \otimes \mathbf{M}^{(N_\mathbf{M})^T} \otimes \mathbf{S}_r^T) \\ &= (\mathbf{S}_l\mathbf{S}_l^T) \otimes (\mathbf{M}^{(N_\mathbf{M})}\mathbf{M}^{(N_\mathbf{M})^T}) \otimes (\mathbf{S}_r\mathbf{S}_r^T) \\ &= \mathbf{I}_L \otimes \mathbf{I}_{N_\mathbf{M}} \otimes \mathbf{I}_R = \mathbf{I}_N,\end{aligned}$$

Hence $\mathbf{V}^{(N)}$, like order-N WHT, is orthogonal and contains N orthonormal basis vectors.

The 1D GCK proposed in [32] can be represented as follows:

$$\mathbf{V}^{(N)} = \mathbf{M}^{(N_\mathbf{M})} \otimes \mathbf{S}_r, \tag{4.14}$$

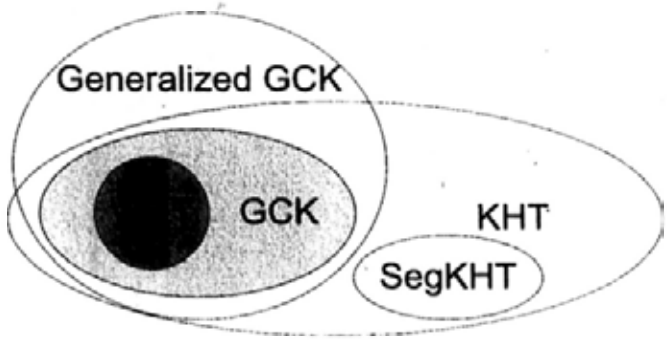


Figure 4.2: *Relationship among WHT, GCK, generalized GCK, SegKHT and KHT. SegKHT, which will be introduced in Section 4.4, is a subset of KHT that cannot be represented by GCK or generalized GCK.*

where $\mathbf{M}^{(N_\mathbf{M})}$ is the $N_\mathbf{M} \times N_\mathbf{M}$ WHT matrix, $\mathbf{S}_r$ is a $R \times R$ matrix and $N = N_\mathbf{M} R$. $\mathbf{S}_r$ is related to the "seed" vectors in [32]. The generalized GCK proposed in [51] is defined as:

$$\mathbf{V}^{(N)} = \left(\otimes_{k=0}^{G-1} \begin{bmatrix} 1 & a_k \\ 1 & -a_k \end{bmatrix} \right) \oslash \mathbf{S}_r, a_k \in \mathbb{Z}^+. \tag{4.15}$$

The relationship among WHT, GCK, generalized GCK and KHT is described in Fig. 4.2. WHT is a member of the GCK with $\mathbf{S}_r = \mathbf{I}_1$ in (4.14). GCK is a subset of the KHT with $\mathbf{S}_l = \mathbf{I}_1$ for KHT matrix $\mathbf{V}^{(N)} = \mathbf{S}_l \otimes \mathbf{M}^{(N_\mathbf{M})} \oslash \mathbf{S}_r$ in (4.4). GCK is also a subset of generalized GCK. Generalized GCK and KHT are different sets of transforms that are both supersets of GCK.

4.2.4 Definition of α-index and Being α^2-related

Let the row index of an order-$N_\mathbf{M}$ WHT matrix $\mathbf{M}^{(N_\mathbf{M})}$ be $i_\mathbf{M}$. The binary sequence α=$[\alpha_{G_\mathbf{M}-1} \ldots \alpha_g \ldots \alpha_1 \ \alpha_0]$ for $\alpha_g \in \{0, 1\}$ and $g = 0, \ldots, G_\mathbf{M} - 1$ is called the α-index of $i_\mathbf{M}$, where

$$i_\mathbf{M} = \alpha_{G_\mathbf{M}-1} 2^{G_\mathbf{M}-1} + \cdots + \alpha_1 2^1 + \alpha_0 2^0, N_\mathbf{M} = 2^{G_\mathbf{M}}. \tag{4.16}$$

Thus the α-index is the binary representation of $i_\mathbf{M}$.

Let $i_\mathbf{M}^{(0)}, i_\mathbf{M}^{(1)}, i_\mathbf{M}^{(2)}$ and $i_\mathbf{M}^{(3)}$ be four possible values of $i_\mathbf{M}$. If the α-indices of $i_\mathbf{M}^{(0)}, i_\mathbf{M}^{(1)}, i_\mathbf{M}^{(2)}$ and $i_\mathbf{M}^{(3)}$ are only different at α_g and α_{g+1}, then we say that: 1) WHT basis vectors

$\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(0)})}$, $\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(1)})}$, $\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(2)})}$ and $\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(3)})}$ are α^2-related at g; 2) the corresponding KHT projection values $y(N,i^{(0)},j)$, $y(N,i^{(1)},j)$, $y(N,i^{(2)},j)$ and $y(N,i^{(3)},j)$ represented as follows are α^2-related at g:

$$\begin{bmatrix} y(N,i^{(0)},j) \\ y(N,i^{(1)},j) \\ y(N,i^{(2)},j) \\ y(N,i^{(3)},j) \end{bmatrix} = \left(\vec{\mathbf{s}}_l^{(i_l)^T} \otimes \begin{bmatrix} \vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(0)})^T} \\ \vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(1)})^T} \\ \vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(2)})^T} \\ \vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(3)})^T} \end{bmatrix} \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \right) \vec{\mathbf{x}}_w^{(N,j)}. \tag{4.17}$$

Note that the basis vectors $\vec{\mathbf{v}}^{(N,i^{(n)})}$ generating $y(N,i^{(n)},j)$ for $n=0,1,2,3$ in (4.17) have the same $\vec{\mathbf{s}}_l^{(i_l)}$ and the same $\vec{\mathbf{s}}_r^{(i_r)}$ but different $\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M}^{(n)})}$. Without losing generality, let the $i_\mathbf{M}^{(0)}, i_\mathbf{M}^{(1)}, i_\mathbf{M}^{(2)}$ and $i_\mathbf{M}^{(3)}$ in (4.17) be represented by:

$$i_\mathbf{M}^{(0)} = \alpha_{G_\mathbf{M}-1}2^{G_\mathbf{M}-1} + \ldots + 0\cdot 2^{g+1} + 0\cdot 2^g + \cdots + \alpha_0 2^0, \tag{4.18}$$

$$i_\mathbf{M}^{(1)} = \alpha_{G_\mathbf{M}-1}2^{G_\mathbf{M}-1} + \ldots + 0\cdot 2^{g+1} + 1\cdot 2^g + \cdots + \alpha_0 2^0, \tag{4.19}$$

$$i_\mathbf{M}^{(2)} = \alpha_{G_\mathbf{M}-1}2^{G_\mathbf{M}-1} + \ldots + 1\cdot 2^{g+1} + 0\cdot 2^g + \cdots + \alpha_0 2^0, \tag{4.20}$$

$$i_\mathbf{M}^{(3)} = \alpha_{G_\mathbf{M}-1}2^{G_\mathbf{M}-1} + \ldots + 1\cdot 2^{g+1} + 1\cdot 2^g + \cdots + \alpha_0 2^0. \tag{4.21}$$

Some WHT basis vectors $\vec{\mathbf{m}}^{(N_\mathbf{M},i_\mathbf{M})}$ and their α-indices for $N_\mathbf{M}=4$, 8 are given in Table 4.1. For example, the α-indices [00], [01], [10] and [11] of 0, 1, 2 and 3 are only different at α_0 and α_1, thus WHT basis vectors $\vec{\mathbf{m}}^{(4,0)}$, $\vec{\mathbf{m}}^{(4,1)}$, $\vec{\mathbf{m}}^{(4,2)}$ and $\vec{\mathbf{m}}^{(4,3)}$ are α^2-related at 0. The α-indices [000], [010], [100] and [110] of 0, 2, 4 and 6 are only different at α_1 and α_2, thus $\vec{\mathbf{m}}^{(8,0)}$, $\vec{\mathbf{m}}^{(8,2)}$, $\vec{\mathbf{m}}^{(8,4)}$ and $\vec{\mathbf{m}}^{(8,6)}$ are α^2-related at 1.

As an example for α^2-related KHT projection values, when $\mathbf{S}_l \in \mathbb{R}^{2\times 2}$, $\mathbf{M}^{(N_\mathbf{M})} = \mathbf{M}^{(4)}$ and $\mathbf{S}_r \in \mathbb{R}^{2\times 2}$, we have:

$$\mathbf{V}^{(16)}\vec{\mathbf{x}}_w^{(N,j)} = \left(\begin{bmatrix} \vec{\mathbf{s}}_l^{(0)^T} \\ \vec{\mathbf{s}}_l^{(1)^T} \end{bmatrix} \otimes \begin{bmatrix} \vec{\mathbf{m}}^{(4,0)^T} \\ \vec{\mathbf{m}}^{(4,1)^T} \\ \vec{\mathbf{m}}^{(4,2)^T} \\ \vec{\mathbf{m}}^{(4,3)^T} \end{bmatrix} \otimes \begin{bmatrix} \vec{\mathbf{s}}_r^{(0)^T} \\ \vec{\mathbf{s}}_r^{(1)^T} \end{bmatrix} \right) \vec{\mathbf{x}}_w^{(N,j)}$$

$$= \begin{bmatrix} \vec{\mathbf{s}}_l^{(0)^T} \otimes \vec{\mathbf{m}}^{(4,0)^T} \otimes \vec{\mathbf{s}}_r^{(0)^T} \\ \vec{\mathbf{s}}_l^{(0)^T} \otimes \vec{\mathbf{m}}^{(4,0)^T} \otimes \vec{\mathbf{s}}_r^{(1)^T} \\ \vec{\mathbf{s}}_l^{(0)^T} \otimes \vec{\mathbf{m}}^{(4,1)^T} \otimes \vec{\mathbf{s}}_r^{(0)^T} \\ \vec{\mathbf{s}}_l^{(0)^T} \otimes \vec{\mathbf{m}}^{(4,1)^T} \otimes \vec{\mathbf{s}}_r^{(1)^T} \\ \vec{\mathbf{s}}_l^{(0)^T} \otimes \vec{\mathbf{m}}^{(4,2)^T} \otimes \vec{\mathbf{s}}_r^{(0)^T} \\ \vec{\mathbf{s}}_l^{(0)^T} \otimes \vec{\mathbf{m}}^{(4,2)^T} \otimes \vec{\mathbf{s}}_r^{(1)^T} \\ \vec{\mathbf{s}}_l^{(0)^T} \otimes \vec{\mathbf{m}}^{(4,3)^T} \otimes \vec{\mathbf{s}}_r^{(0)^T} \\ \vec{\mathbf{s}}_l^{(0)^T} \otimes \vec{\mathbf{m}}^{(4,3)^T} \otimes \vec{\mathbf{s}}_r^{(1)^T} \\ \vdots \end{bmatrix} \vec{\mathbf{x}}_w^{(N,j)} = \begin{bmatrix} y(16,0,j) \\ y(16,1,j) \\ y(16,2,j) \\ y(16,3,j) \\ y(16,4,j) \\ y(16,5,j) \\ y(16,6,j) \\ y(16,7,j) \\ \vdots \end{bmatrix}, \tag{4.22}$$

Basis vector	$[1\ 1\ 1\ 1]^T$	$[1\ 1\ -1\ -1]^T$	$[1\ -1\ 1\ -1]^T$	$[1\ -1\ -1\ 1]^T$
$\vec{\mathbf{m}}^{(N_M, i_M)}$	$\vec{\mathbf{m}}^{(4,0)}$	$\vec{\mathbf{m}}^{(4,1)}$	$\vec{\mathbf{m}}^{(4,2)}$	$\vec{\mathbf{m}}^{(4,3)}$
α-index	[00]	[01]	[10]	[11]

Basis vector	$[1\ 1\ 1\ 1\ 1\ 1\ 1\ 1]^T$	$[1\ 1\ 1\ 1\ -1\ -1\ -1\ -1]^T$	...
$\vec{\mathbf{m}}^{(N_M, i_M)}$	$\vec{\mathbf{m}}^{(8,0)}$	$\vec{\mathbf{m}}^{(8,1)}$	...
α-index	[000]	[001]	...

Table 4.1: *The α-index for dyadic-ordered WHT basis vector.*

	Meanings
$\dot{\mathbf{x}}_w^{(N,j)}$	The input window starting at x_j having length N: $[x_j, x_{j+1}, \ldots, x_{j+N-1}]^T$.
$\mathbf{M}^{(N_M)}$	1D $N_M \times N_M$ WHT matrix.
$\dot{\mathbf{m}}^{(N_M, i_M)}$	The i_Mth WHT basis vector having length N_M.
$\mathbf{V}^{(N)}$	1D KHT matrix $\mathbf{V}^{(N)} = \mathbf{S}_l \otimes \mathbf{M}^{(N_M)} \otimes \mathbf{S}_r$, where $\mathbf{S}_l$ and $\mathbf{S}_r$ are $L \times L$ and $R \times R$ matrices respectively.
$\dot{\mathbf{v}}^{(N,i)}$	The ith KHT basis vector having length N.
$\mathbf{V}_s^{(N)}$	The SegKHT matrix $\mathbf{V}_s^{(N)} = \mathbf{I}_{L_s} \otimes \mathbf{V}^{(N_s)}$.
$y(N, i, j)$	$y(N, i, j) = \dot{\mathbf{v}}^{(N,i)^T} \dot{\mathbf{x}}_w^{(N,j)}$. The ith order-N KHT projection value at the jth window.
α-index of i_M	Sequence $[a_{G_M-1} \ldots a_g \ldots a_1\ a_0]$, which is the binary representation of i_M: $i_M = a_{G_M-1} 2^{G_M-1} + \ldots + a_0 2^0$.
$\dot{\mathbf{m}}^{(N_M, i_M^{(0)})}$, $\dot{\mathbf{m}}^{(N_M, i_M^{(1)})}$, $\dot{\mathbf{m}}^{(N_M, i_M^{(2)})}$ and $\dot{\mathbf{m}}^{(N_M, i_M^{(3)})}$ are α^2-related at g	The α-indices of $i_M^{(0)}$, $i_M^{(1)}$, $i_M^{(2)}$ and $i_M^{(3)}$ are only different at a_g and a_{g+1}.

Table 4.2: *Symbols and terms defined for KHT. The SegKHT matrix* $\mathbf{V}_s^{(N)}$ *will be defined in Section 4.4.*

where WHT basis vectors $\vec{\mathbf{m}}^{(4,0)}$, $\vec{\mathbf{m}}^{(4,1)}$, $\vec{\mathbf{m}}^{(4,2)}$ and $\vec{\mathbf{m}}^{(4,3)}$ are α^2-related at 0. $y(16,0,j)$, $y(16,2,j)$, $y(16,4,j)$ and $y(16,6,j)$ in (4.22) are α^2-related since they can be represented as follows:

$$\begin{bmatrix} y(16,0,j) \\ y(16,2,j) \\ y(16,4,j) \\ y(16,6,j) \end{bmatrix} = \left(\vec{\mathbf{s}}_l^{(0)^T} \otimes \begin{bmatrix} \dot{\mathbf{m}}^{(4,0)^T} \\ \dot{\mathbf{m}}^{(4,1)^T} \\ \dot{\mathbf{m}}^{(4,2)^T} \\ \dot{\mathbf{m}}^{(4,3)^T} \end{bmatrix} \otimes \mathbf{s}_r^{(0)^T} \right) \dot{\mathbf{x}}_w^{(N,j)}, \tag{4.23}$$

where $i^{(0)}$=0, $i^{(1)}$=2, $i^{(2)}$=4, $i^{(3)}$=6, $i_M^{(0)}$=0, $i_M^{(1)}$=1, $i_M^{(2)}$=2, $i_M^{(3)}$=3 for (4.17)-(4.21). Similarly, $y(16,1,j)$, $y(16,3,j)$, $y(16,5,j)$ and $y(16,7,j)$ in (4.22) are α^2-related. Table 4.2 summarizes the main definitions used for KHT.

4.3 The Fast KHT Algorithm

This section gives an example of computing 1D order-4 WHT on sliding windows using the proposed KHT algorithm in Section 4.3.1, and then describes the KHT algorithm for 1D order-N KHT and high dimensional KHT.

4.3.1 The Fast KHT Algorithm for Order-4 WHT

If $\mathbf{S}_l = \mathbf{S}_r = \mathbf{I}_1$, then KHT matrix $\mathbf{V}^{(N)}$ is the order-N WHT matrix $\mathbf{M}^{(N)}$. When $N = 4$, we have:

$$\vec{\mathbf{y}}^{(4,j)} = \begin{bmatrix} y(4,0,j) \\ y(4,1,j) \\ y(4,2,j) \\ y(4,3,j) \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_j \\ x_{j+1} \\ x_{j+2} \\ x_{j+3} \end{bmatrix}. \tag{4.24}$$

Let $\Delta(4,j) = y(4,0,j) - y(4,0,j+1)$. Equation (4.24) implies that:

$$\begin{aligned} &\begin{bmatrix} y(4,0,j) - y(4,0,j+1) \\ y(4,3,j) + y(4,1,j+1) \\ y(4,2,j) + y(4,2,j+1) \\ y(4,1,j) - y(4,3,j+1) \end{bmatrix} \\ &= \frac{1}{2} \begin{bmatrix} (x_j + x_{j+1} + x_{j+2} + x_{j+3}) - (x_{j+1} + x_{j+2} + x_{j+3} + x_{j+4}) \\ (x_j - x_{j+1} - x_{j+2} + x_{j+3}) + (x_{j+1} + x_{j+2} - x_{j+3} - x_{j+4}) \\ (x_j - x_{j+1} + x_{j+2} - x_{j+3}) + (x_{j+1} - x_{j+2} + x_{j+3} - x_{j+4}) \\ (x_j + x_{j+1} - x_{j+2} - x_{j+3}) - (x_{j+1} - x_{j+2} - x_{j+3} + x_{j+4}) \end{bmatrix} \\ &= \frac{1}{2} \begin{bmatrix} x_j - x_{j+4} \\ x_j - x_{j+4} \\ x_j - x_{j+4} \\ x_j - x_{j+4} \end{bmatrix} = \begin{bmatrix} y(4,0,j) - y(4,0,j+1) \\ y(4,0,j) - y(4,0,j+1) \\ y(4,0,j) - y(4,0,j+1) \\ y(4,0,j) - y(4,0,j+1) \end{bmatrix} = \begin{bmatrix} \Delta(4,j) \\ \Delta(4,j) \\ \Delta(4,j) \\ \Delta(4,j) \end{bmatrix} \end{aligned} \tag{4.25}$$

Hence, $\Delta(4,j)$ relates the projection values in window j and $j+1$. Equation (4.25) implies

$$\begin{bmatrix} y(4,1,j+1) \\ y(4,2,j+1) \\ y(4,3,j+1) \end{bmatrix} = \begin{bmatrix} -y(4,3,j) \\ -y(4,2,j) \\ y(4,1,j) \end{bmatrix} + \begin{bmatrix} \Delta(4,j) \\ \Delta(4,j) \\ -\Delta(4,j) \end{bmatrix}. \tag{4.26}$$

When we compute the projection values in window $j+1$ on sliding windows, the projection values in windows 0, 1, ..., j have been computed. Given $y(4,0,j+1)$, the KHT algorithm: 1) computes $\Delta(4,j)$ as $\Delta(4,j) = y(4,0,j) - y(4,0,j+1)$ by 1 addition;

2) uses $\Delta(4,j)$ and projection values in window j for computing the 3 projection values in window $j+1$ by 3 additions using (4.26).

The GCK algorithm in [32] computes the WHT projection values in window $j+1$ from the projection values in window j and $j-1$ as follows:

$$y(4,1,j+1)=y(4,0,j-1)-y(4,0,j+1)-y(4,1,j-1), \tag{4.27}$$

$$y(4,3,j+1)=y(4,1,j)-y(4,1,j+1)-y(4,3,j), \tag{4.28}$$

$$y(4,2,j+1)=y(4,2,j-1)-y(4,3,j-1)-y(4,3,j+1). \tag{4.29}$$

Given $y(4,0,j+1)$, the GCK algorithm will: 1) compute the 1st projection value, i.e. $y(4,1,j+1)$, from the 0th projection values $y(4,0,j-1)$ and $y(4,0,j+1)$ by 2 additions in (4.27); 2) compute the 3rd projection value from the 1st projection values by 2 additions in (4.28); 3) compute the 2nd projection value from the 3rd projection values by 2 additions in (4.29).

4.3.2 The Fast KHT Algorithm for Order-N KHT

The theorem below is proved in the appendix.

Theorem 4.2 If four order-N KHT projection values $y(N,i^{(n)},j)=[\vec{\mathbf{s}}_l^{(i_l)}\otimes\vec{\mathbf{m}}^{(N_{\mathbf{M}},i_{\mathbf{M}}^{(n)})}\otimes\vec{\mathbf{s}}_r^{(i_r)}]^T\vec{\mathbf{x}}_w^{(N,j)}$ for $n=0,\ldots,3$ as defined in (4.9) are α^2-related at g, where $i_{\mathbf{M}}^{(n)}$ are represented in (4.18)-(4.21), $\vec{\mathbf{s}}_l^{(i_l)}$ has length L and $\vec{\mathbf{s}}_r^{(i_r)}$ has length R, then we have:

$$\begin{bmatrix} y(N,i^{(1)},j+R_4)\\ y(N,i^{(2)},j+R_4)\\ y(N,i^{(3)},j+R_4)\end{bmatrix}=\begin{bmatrix} -y(N,i^{(3)},j)\\ -y(N,i^{(2)},j)\\ y(N,i^{(1)},j)\end{bmatrix}+\begin{bmatrix} \Delta(N,j,R_4)\\ \Delta(N,j,R_4)\\ -\Delta(N,j,R_4)\end{bmatrix}, \tag{4.30}$$

$$\text{where}\quad R_4=2^{G_{\mathbf{M}}-g-2}R, \tag{4.31}$$

$$\Delta(N,j,R_4)=y(N,i^{(0)},j)-y(N,i^{(0)},j+R_4). \tag{4.32}$$

Table 4.3 shows the steps and corresponding number of additions required using Theorem 4.2 to compute the 3 projection values $y(N,i^{(n)},j+R_4)$ for $n=1,2,3$ from $y(N,i^{(0)},j+R_4)$. The computation in (4.30) corresponds to the example in (4.26), where N=4, $i^{(0)}$=0, $i^{(1)}$=1, $i^{(2)}$=2, $i^{(3)}$=3, $y(4,i,j)$ for $i=0,1,2,3$ are α^2-related at 0 (thus $g=0$), $G_{\mathbf{M}}=2$, $R=1$, so $R_4=2^{2-0-2}\cdot 1=1$ and $\Delta(N,j,R_4)=\Delta(4,j)$.

Theorem 4.2 shows how to compute the other 3 projection values from $y(N,i^{(0)},j+R_4)$. The following corollary shows that we can compute $y(N,i^{(0)},j+R_4)$ from one of

Step a $y(N, i^{(0)}, j+R_4)$ is provided by other approaches. - The computation required is not counted. ***Step b*** $\Delta(N, j, R_4) = y(N, i^{(0)}, j) - y(N, i^{(0)}, j+R_4)$ - One addition is required. ***Step c*** $y(N, i^{(1)}, j+R_4) = \Delta(N, j, R_4) - y(N, i^{(3)}, j)$, $y(N, i^{(2)}, j+R_4) = \Delta(N, j, R_4) - y(N, i^{(2)}, j)$, $y(N, i^{(3)}, j+R_4) = y(N, i^{(1)}, j) - \Delta(N, j, R_4)$. - Three additions are required. Note that $y(N, i^{(n)}, j)$ for n=0, ..., 3 are obtained during previous computation.

Table 4.3: *Computation of order-N KHT using the KHT algorithm*

the other 3 projection values:

Corollary 4.3

$$y(N, i^{(0)}, j + R_4) = y(N, i^{(0)}, j) - \Delta(N, j, R_4), \tag{4.33}$$

$$\begin{aligned} \Delta(N, j, R_4) &= y(N, i^{(3)}, j) + y(N, i^{(1)}, j + R_4) \\ &= y(N, i^{(2)}, j) + y(N, i^{(2)}, j + R_4) \\ &= y(N, i^{(1)}, j) - y(N, i^{(3)}, j + R_4). \end{aligned} \tag{4.34}$$

Equation (4.33) is derived from (4.32); (4.34) is from (4.30). The computation of $\vec{\mathbf{y}}^{(N,j)}$ using (4.30)-(4.34) is named as the fast KHT algorithm. Here is a summary of the fast KHT algorithm. If one of the 4 α^2-related projection values $y(N, i^{(n)}, j + R_4)$ for $n = 0, \ldots, 3$ is provided, then we can: 1) use this provided projection value to obtain $\Delta(4, j, R_4)$ by 1 addition using (4.32) or (4.34); 2) obtain the other 3 projection values by 3 additions using (4.30), (4.33). Therefore, the KHT algorithm requires 4 additions for obtaining 3 projection values, i.e. 4/3 additions per window per projection value independent of KHT size N. In comparison, the GCK algorithm requires 2 additions per projection value and the algorithm in [33] requires 3/2 additions.

The GCK algorithm, the algorithm in [33] and the KHT algorithm assume that one projection value is provided on all window positions. Let the number of additions per window required for computing this projection value be B_1. The GCK algorithm requires $2(u-1) + B_1$ additions per window for obtaining u projection values and the KHT algorithm requires $4(u-1)/3 + B_1$ additions. Normally, this provided projection value is the *dc* component for 2D WHT and can be computed using box-technique [47] by $B_1 = 4$ additions per window or using integral image [48] by $B_1 = 3$ additions per

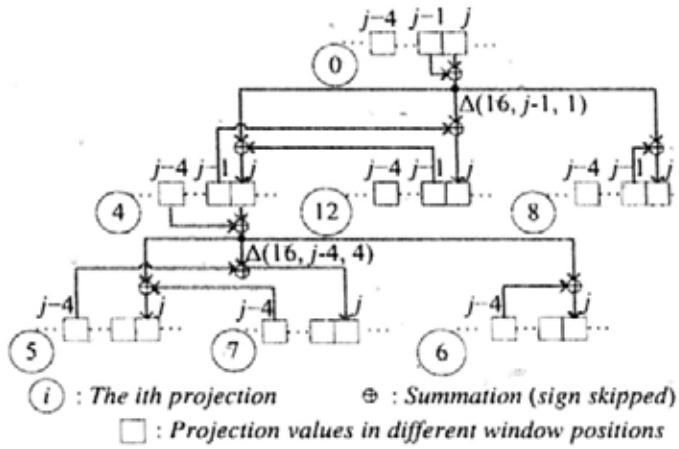


Figure 4.3: *Utilization of the KHT algorithm for obtaining the other projection values from the 0th projection value. The number i in circle denotes the ith projection. The rectangles denote projection values in different window positions. The signs are skipped for the summation operations in this figure.*

window. Since B_1 is a constant and is relatively small compared with the computation required by u when u is large, B_1 is skipped in [32] and in the followings of this chapter.

Generally, if we are provided with one of the $N_{\mathbf{M}}$ projection values in window j, i.e. $y(N,i,j) = [\vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(N_{\mathbf{M}},i_{\mathbf{M}})} \otimes \vec{\mathbf{s}}_r^{(i_r)}]^T \vec{\mathbf{x}}_w^{(N,j)}$ for $i_{\mathbf{M}} = 0,\ldots,N_{\mathbf{M}}-1$, then we can efficiently compute the remaining $N_{\mathbf{M}}-1$ projection values. As a special case, when $\vec{\mathbf{s}}_l = \mathbf{I}_1$, $\vec{\mathbf{s}}_r = \mathbf{I}_1$, KHT is WHT and we have $L = R = 1$. Using order-16 WHT as an example, we show in Fig. 4.3 how to obtain all projection values in window j with the 0th projection value provided:

1) Since the 0th, 4th, 8th and 12th WHT projection values having α-indices [0000], [0100], [1000] and [1100] respectively are α^2-related at 2, i.e. $g = 2$, the KHT algorithm computes the $y(16,4,j)$, $y(16,8,j)$ and $y(16,12,j)$ from the $y(16,0,j)$, which is shown in Fig. 4.3. In this case, we have $g = 2$ and $R = 1$ for these 4 WHT projection values, $N = 16$ and $G_{\mathbf{M}} = 4$ for order-16 WHT, so we have the followings from Theorem 4.2:

$$\begin{bmatrix} y(16,4,j) \\ y(16,8,j) \\ y(16,12,j) \end{bmatrix} = \begin{bmatrix} -y(16,12,j-1) \\ -y(16,8,j-1) \\ y(16,4,j-1) \end{bmatrix} + \begin{bmatrix} \Delta(16,j-1,1) \\ \Delta(16,j-1,1) \\ -\Delta(16,j-1,1) \end{bmatrix},$$

$$\text{where} \quad R_4 = 2^{G_{\mathbf{M}}-g-2}R = 1^{4-2-2} = 1,$$

$$\Delta(16,j-1,1) = y(16,0,j-1) - y(16,0,j). \tag{4.35}$$

2) The projection value $y(16,4,j)$ obtained in the previous step can be used to obtain the projection values $y(16,5,j)$, $y(16,6,j)$ and $y(16,7,j)$ because these 4 projection values are α^2-related at $g=0$, which is shown in Fig. 4.3. In this case, we have $g=0$, $R=1$, $N=16$, $G_{\mathbf{M}}=4$ and the followings from Theorem 4.2:

$$\begin{bmatrix} y(16,5,j) \\ y(16,6,j) \\ y(16,7,j) \end{bmatrix} = \begin{bmatrix} -y(16,7,j-4) \\ -y(16,6,j-4) \\ y(16,5,j-4) \end{bmatrix} + \begin{bmatrix} \Delta(16,j-4,4) \\ \Delta(16,j-4,4) \\ -\Delta(16,j-4,4) \end{bmatrix},$$

where $R_4 = 2^{G_{\mathbf{M}}-g-2}R = 2^{4-0-2} = 4$,

$$\Delta(16,j-4,4) = y(16,4,j-4) - y(16,4,j). \tag{4.36}$$

3) The remaining projection values are similarly obtained.

The GCK algorithm in [32] is specific for GCK. In the appendix, we prove that the GCK algorithm in [32] is also applicable for KHT.

4.3.3 High Dimensional KHT

In this section, we skip the size denotation of matrices for simplicity. D dimensional separable KHT can be represented using Kronecker product as follows:

$$\begin{aligned} &[\mathbf{V}_1 \otimes \cdots \otimes \mathbf{V}_d \otimes \ldots \otimes \mathbf{V}_D] vec(X) \\ &= [\mathbf{V}_1 \otimes \cdots \otimes (\mathbf{S}_{l_d} \otimes \mathbf{M}_d \otimes \mathbf{S}_{r_d}) \otimes \cdots \mathbf{V}_D] vec(X), \end{aligned} \tag{4.37}$$

where $\mathbf{V}_d = \mathbf{S}_{l_d} \otimes \mathbf{M}_d \otimes \mathbf{S}_{r_d}$ for $d = 1, \ldots, D$ is the KHT matrix in dimension d, $\mathbf{S}_{l_d}$ is the left matrix, $\mathbf{M}_d$ is the WHT matrix, $\mathbf{S}_{r_d}$ is the right matrix and $vec(X)$ is the D dimensional input X that is arranged as a 1D vector.

We have illustrated the KHT algorithm in 1D case where the input window is a vector. The KHT algorithm can be used for computing KHT of higher dimensions, e.g. 2D for images. And the computation required is 4/3 additions per projection value per window independent of dimension and size. Detailed information is provided in Appendix C.

4.3.4 Ordering KHT Projection Values

The Gray-code sequence (GCS) is proposed in [32] to order the GCK projection values so that the current projection value can be efficiently computed from the previously computed projection values using the GCK algorithm. To order 2D projection values,

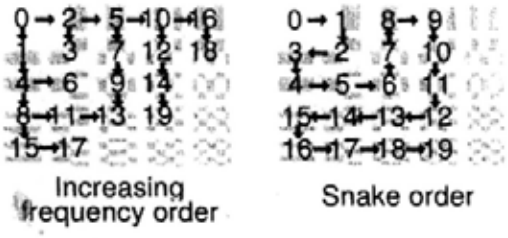


Figure 4.4: *Snake order and increasing frequency order proposed in [2] for WHT. Numbers denote the order. Arrows denote the computation dependence. Projection value 1 is computed from 0 in both orders.*

Moshe and Hel-Or propose two orders for computing GCK on images in [2]. They are the snake order and increasing frequency order. Fig. 4.4 shows the two orders for the first 20 2D WHT projection values. The increasing frequency order is more efficient in packing energy while the snake order is more efficient in memory requirement.

For efficient computation of 1D KHT, the KHT projection values can be arranged in sequency order, i.e. in ascending order of the number of zero crossings of the basis vectors. To efficiently compute projection values using the KHT algorithm for 2D natural images, we can use the order shown in Fig. 4.5. For this order, 16 projection values form a group. Groups are arranged in an increasing frequency order and projection values within a group have a fixed order. This order for larger sizes of WHT or KHT can be similarly obtained. In the experimental results, we show that the KHT algorithm using this order is faster than the GCK algorithm using the increasing frequency order.

When specific projection values are required, we recommend constructing the GCS first and then examining the α^2-related projection values in the GCS for using the KHT algorithm.

Figure 4.5: *The ordering 2D 8 × 8 WHT for fast KHT algorithm. Numbers denote the order. Solid arrows denote the computation dependence using the KHT algorithm while dashed arrows denote the computation dependence using the GCK algorithm. For example, projection values 4, 5 and 6 are computed from 0 by the KHT algorithm while 32 is computed from 6 by the GCK algorithm.*

4.4 The Segmented KHT

4.4.1 The Definition of Segmented KHT

Define the order-N Segmented KHT (SegKHT) matrix $\mathbf{V}_s^{(N)}$ of identity matrix $\mathbf{I}_{L_s}$ and KHT matrix $\mathbf{V}^{(N_s)}$ as follows:

$$\begin{aligned}\mathbf{V}_s^{(N)} &= [\vec{\mathbf{v}}_s^{(N,0)}, \ldots, \vec{\mathbf{v}}_s^{(N,i)}, \ldots, \vec{\mathbf{v}}_s^{(N,N-1)}]^T \\ &= \mathbf{I}_{L_s} \otimes \mathbf{V}^{(N_s)} \\ &= \begin{bmatrix} \mathbf{V}^{(N_s)} & 0 & 0 & 0 \\ 0 & \mathbf{V}^{(N_s)} & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \mathbf{V}^{(N_s)} \end{bmatrix}_{L_s \times L_s}\end{aligned} \tag{4.38}$$

where $L_s = 2, 3, 4, \ldots$ is the segmentation parameter, $N = L_s N_s$, $\vec{\mathbf{v}}_s^{(N,i)}$ for $i = 0, \ldots, N-1$ is the ith SegKHT basis vector having length N and subscript $\cdot_s$ is used for denotation related to SegKHT. The order-N SegKHT matrix is an $N \times N$ matrix. Property 4.1 shows how to make the KHT matrix $\mathbf{V}^{(N_s)}$ in (4.38) orthogonal. And it is easy to see from (4.38) that $\mathbf{V}_s^{(N)}$ is orthogonal if $\mathbf{V}^{(N_s)}$ is orthogonal. Thus we can make $\mathbf{V}^{(N_s)}$ orthogonal to obtain orthogonal SegKHT matrix and apply SegKHT for transform domain pattern matching introduced in Section 1.2.4.

The SegKHT matrix defined in (4.38) can be considered as the KHT matrix defined

in (4.4) as follows:

$$\begin{aligned}\mathbf{V}_s^{(N)} &= \mathbf{I}_{L_s} \otimes \mathbf{V}^{(N_s)} \\ &= \mathbf{I}_{L_s} \otimes (\mathbf{S}_l \otimes \mathbf{M}^{(N_M)} \otimes \mathbf{S}_r) \\ &= (\mathbf{I}_{L_s} \otimes \mathbf{S}_l) \otimes \mathbf{M}^{(N_M)} \otimes \mathbf{S}_r \\ &= \mathbf{S}_{l1} \otimes \mathbf{M}^{(N_M)} \otimes \mathbf{S}_r.\end{aligned} \tag{4.39}$$

This KHT matrix is the Kronecker product of left matrix $\mathbf{S}_{l1} = (\mathbf{I}_{L_s} \otimes \mathbf{S}_l)$, WHT matrix $\mathbf{M}^{(N_M)}$ and right matrix $\mathbf{S}_r$. Thus SegKHT is a subset of KHT. KHT is used for designing new transforms and the fast KHT algorithm; SegKHT is a subset of KHT that has accelerated computation based on the proposed fast KHT algorithm. However, SegKHT cannot be represented by GCK matrix that is in the form of $\mathbf{V}^{(N)} = \mathbf{M}^{(N_M)} \otimes \mathbf{S}_r$. The relationship among GCK, KHT and SegKHT is described in Fig. 4.2.

Let $\vec{\mathbf{y}}_s^{(N,j)}$ for $j = 0, \ldots, W-1$ be the SegKHT projection value vector containing all projection values at the jth window and

$$\begin{aligned}\vec{\mathbf{y}}_s^{(N,j)} &= \mathbf{V}_s^{(N)} \vec{\mathbf{x}}_w^{(N,j)} \\ &= [y_s(N,0,j), \ldots, y_s(N,i,j) \ldots, y_s(N,N-1,j)]^T,\end{aligned} \tag{4.40}$$

where $y_s(N,i,j) = \vec{\mathbf{v}}_s^{(N,i)^T} \vec{\mathbf{x}}_w^{(N,j)}$ is the ith SegKHT projection value for the jth window.

When the KHT matrix $\mathbf{V}^{(N_s)}$ of SegKHT matrix is a WHT matrix $\mathbf{M}^{(N_s)}$, we call it the Segmented WHT (SegWHT). For example, when $L_s = 2, N_s = 4, N = 8$ and $\mathbf{V}^{(N_s)} = \mathbf{M}^{(4)}$ in (4.38), we have an order-8 SegWHT matrix as follows:

$$\mathbf{V}_s^{(8)} = \mathbf{I}_2 \otimes \mathbf{M}^{(4)} = \begin{bmatrix} \mathbf{M}^{(4)} & 0 \\ 0 & \mathbf{M}^{(4)} \end{bmatrix}. \tag{4.41}$$

4.4.2 Fast Segmented KHT Algorithm

The jth input window of length N can be represented by L_s subwindows of length N_s as follows:

$$\vec{\mathbf{x}}_w^{(N,j)} = \begin{bmatrix} x_j \\ x_{j+1} \\ x_{j+2} \\ \vdots \\ x_{j+N-1} \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{x}}_w^{(N_s,j)} \\ \vec{\mathbf{x}}_w^{(N_s,j+N_s)} \\ \vec{\mathbf{x}}_w^{(N_s,j+2\cdot N_s)} \\ \vdots \\ \vec{\mathbf{x}}_w^{(N_s,j+(L_s-1)\cdot N_s)} \end{bmatrix}, \tag{4.42}$$

From (4.38), (4.40) and (4.42), the SegKHT projection value vector can be represented by L_s KHT projection value vectors as follows:

$$\begin{aligned} \vec{\mathbf{y}}_s^{(N,j)} &= \mathbf{V}_s^{(N)} \vec{\mathbf{x}}_w^{(N,j)} \\ &= \begin{bmatrix} \mathbf{V}^{(N_s)} & 0 & 0 & 0 \\ 0 & \mathbf{V}^{(N_s)} & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathbf{V}^{(N_s)} \end{bmatrix} \begin{bmatrix} \vec{\mathbf{x}}_w^{(N_s,j)} \\ \vec{\mathbf{x}}_w^{(N_s,j+N_s)} \\ \vdots \\ \vec{\mathbf{x}}_w^{(N_s,j+(L_s-1)\cdot N_s)}) \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{V}^{(N_s)} \vec{\mathbf{x}}_w^{(N_s,j)} \\ \mathbf{V}^{(N_s)} \vec{\mathbf{x}}_w^{(N_s,j+N_s)} \\ \vdots \\ \mathbf{V}^{(N_s)} \vec{\mathbf{x}}_w^{(N_s,j+(L_s-1)\cdot N_s)}) \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{y}}^{(N_s,j)} \\ \vec{\mathbf{y}}^{(N_s,j+N_s)} \\ \vdots \\ \vec{\mathbf{y}}^{(N_s,j+(L_s-1)\cdot N_s)} \end{bmatrix}, \end{aligned} \tag{4.43}$$

where $\vec{\mathbf{x}}_w^{(N,j)}$ is the jth input window having length $N(= L_s N_s)$ and $\vec{\mathbf{x}}_w^{(N_s,j)}$ is the jth input window having length N_s; $\vec{\mathbf{y}}_s^{(N,j)}$ is the SegKHT projection value vector at the jth window and $\vec{\mathbf{y}}^{(N_s,j)}$ is the KHT projection value vector at the jth window. In (4.43), we segment the length-N input window $\vec{\mathbf{x}}_w^{(N,j)}$ into L_s length-N_s subwindows $\vec{\mathbf{x}}_w^{(N_s,j)}, \dots, \vec{\mathbf{x}}_w^{(N_s,j+(L_s-1)N_s)}$ and then do order-N_s KHT on these L_s subwindows. This procedure is shown in Fig. 4.6. In this procedure, the projection for a window is decomposed into projection for its subwindows.

According to the result in (4.43), obtaining the SegKHT projection value vector $\vec{\mathbf{y}}_s^{(N,j)}$ in the jth window is equivalent to obtaining the L_s KHT projection value vectors $\vec{\mathbf{y}}^{(N_s,j+nN_s)}$ for $n = 0, \dots, L_s - 1$. Similarly, obtaining $\vec{\mathbf{y}}_s^{(N,j+N_s)}$ in the $j+N_s$th window is equivalent to obtaining L_s vectors $\vec{\mathbf{y}}^{(N_s,j+nN_s)}$ for $n = 1, \dots, L_s$, where the $L_s - 1$ vectors $\vec{\mathbf{y}}^{(N_s,j+nN_s)}$ for $n = 1, \dots, L_s - 1$ have been computed previously in $\vec{\mathbf{y}}_s^{(N,j)}$ and only $\vec{\mathbf{y}}^{(N_s,j+L_sN_s)}$ is not computed previously. Fig. 4.7 shows the relationship between $\vec{\mathbf{y}}_s^{(N,j)}$ and $\vec{\mathbf{y}}_s^{(N,j+N_s)}$. When we compute $\vec{\mathbf{y}}_s^{(N,j+N_s)}$, we need only obtain $\vec{\mathbf{y}}^{(N_s,j+L_sN_s)}$

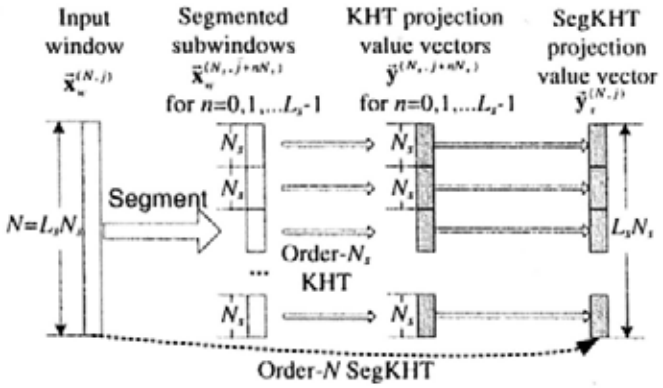


Figure 4.6: *The order-N SegKHT that can be computed by segmenting input window into L_s subwindows having length N_s and then doing order-N_s KHT on the L_s subwindows.*

by about $4N_s/3$ additions using the KHT algorithm. Thus the fast SegKHT algorithm requires about $4N_s/3$ additions to obtain $N(= L_sN_s)$ SegKHT projection values per window, while the GCK algorithm requires $2N$ additions and the algorithm in [33] requires $3N/2$ additions for obtaining N GCK projection values. On average, the fast SegKHT algorithm requires $4/(3L_s)$ addition(s) per projection value.

As an example, we have the followings for the order-8 SegWHT in (4.41):

$$\begin{aligned}\vec{\mathbf{y}}_s^{(8,j)} &= \mathbf{V}_s^{(8)}\vec{\mathbf{x}}_w^{(8,j)} = (\mathbf{I}_2 \otimes \mathbf{M}^{(4)})\vec{\mathbf{x}}_w^{(8,j)} \\ &= \begin{bmatrix} \mathbf{M}^{(4)} & 0 \\ 0 & \mathbf{M}^{(4)} \end{bmatrix} \begin{bmatrix} \vec{\mathbf{x}}_w^{(4,j)} \\ \vec{\mathbf{x}}_w^{(4,j+4)} \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{y}}^{(4,j)} \\ \vec{\mathbf{y}}^{(4,j+4)} \end{bmatrix}\end{aligned} \tag{4.44}$$

$$\begin{aligned}\vec{\mathbf{y}}_s^{(8,j+4)} &= \mathbf{V}_s^{(8)}\vec{\mathbf{x}}_w^{(8,j+4)} = (\mathbf{I}_2 \otimes \mathbf{M}^{(4)})\vec{\mathbf{x}}_w^{(8,j+4)} \\ &= \begin{bmatrix} \mathbf{M}^{(4)} & 0 \\ 0 & \mathbf{M}^{(4)} \end{bmatrix} \begin{bmatrix} \vec{\mathbf{x}}_w^{(4,j+4)} \\ \vec{\mathbf{x}}_w^{(4,j+8)} \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{y}}^{(4,j+4)} \\ \vec{\mathbf{y}}^{(4,j+8)} \end{bmatrix}\end{aligned} \tag{4.45}$$

The relationships in (4.44) and (4.45) are examples for (4.43), where $\mathbf{V}^{(N_s)} = \mathbf{M}^{(4)}$, $L_s = 2, N_s = 4$ and $N = 8$. According to the result in (4.44) and (4.45), obtaining $\vec{\mathbf{y}}_s^{(8,j)}$ is equivalent to obtaining $\vec{\mathbf{y}}^{(4,j)}$ and $\vec{\mathbf{y}}^{(4,j+4)}$. Similarly, obtaining $\vec{\mathbf{y}}_s^{(8,j+4)}$ is equivalent to obtaining $\vec{\mathbf{y}}^{(4,j+4)}$ and $\vec{\mathbf{y}}^{(4,j+8)}$, where $\vec{\mathbf{y}}^{(4,j+4)}$ has been obtained in $\vec{\mathbf{y}}_s^{(8,j)}$. This example is shown in Fig. 4.6. If we compute $\vec{\mathbf{y}}_s^{(8,j+4)}$ on sliding windows: 1) its 4 elements in vector $\vec{\mathbf{y}}^{(4,j+4)}$ have been computed when we compute $\vec{\mathbf{y}}_s^{(8,j)}$; 2) its other 4 elements in $\vec{\mathbf{y}}^{(4,j+8)}$ can be computed by about $16/3(= 4 \cdot 4/3)$ additions using the KHT algorithm. Thus the fast SegKHT algorithm requires about 16/3 additions for obtaining the 8 projection values in $\vec{\mathbf{y}}_s^{(8,j+4)}$, i.e. 2/3 additions per projection value on

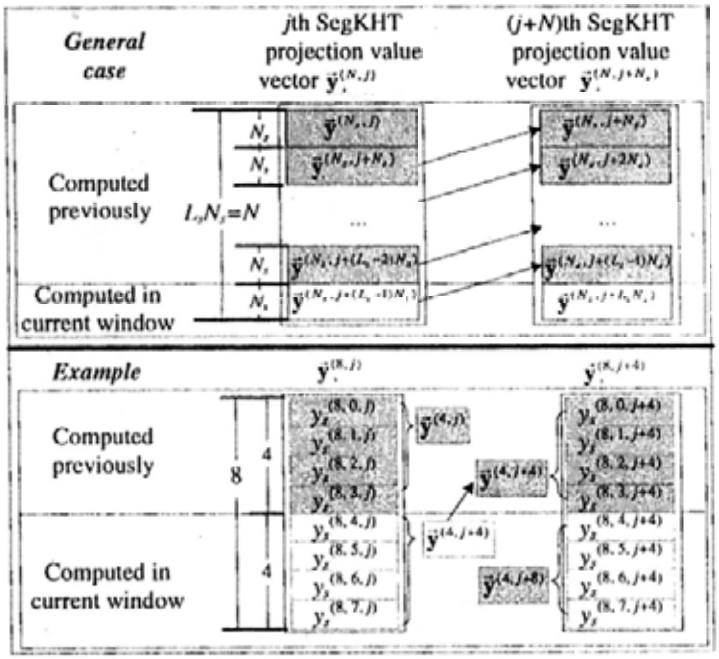


Figure 4.7: *Computing order N SegKHT on sliding windows. For general case, SegKHT projection value vectors* $\vec{\mathbf{y}}_s^{(N,j)}$ *and* $\vec{\mathbf{y}}_s^{(N,j+N_s)}$ *share* L_s-1 *KHT projection value vectors. Arrows in the figure point out the* L_s-1 *shared KHT projection value vectors. For the example in (4.41), we have* $L_s=2, N_s=4$ *and* $N=8$.

average. The KHT algorithm requires 4/3 additions per projection value.

Since the memory storing $\vec{\mathbf{y}}^{(N_s,j)}$ are regarded as SegKHT projection values at L_s different input window positions, $\vec{\mathbf{y}}^{(N_s,j)}$ will be accessed multiple times, which improves memory utilization and saves memory access time when these data are found multiple times in the data cache.

For input window having length N, the L_s basis vectors proposed in [30] can be represented by SegKHT basis vectors while SegKHT contains the other $N-L_s$ basis vectors that cannot be represented in [30]. The method in [35] segments non-rectangular patterns into certain number of rectangles aiming at dealing with non-rectangular pattern matching; the SegKHT segments rectangular pattern into L_s rectangles aiming at improving the computational efficiency. The SegKHT is inspired by the Incremental Dissimilarity Approximations (IDA) algorithm [31], in which Tombari *et al.* achieve computational efficiency by segmenting input data into several parts. The IDA algorithm determines a succession of rejection conditions characterized by increasing rejection ability and computational complexity. The differences between SegKHT and IDA are as follows: 1) the IDA is not transform domain pattern matching algorithm while SegKHT is used for transform domain pattern matching; 2) the IDA segments

input window into subwindows and uses the triangular inequality on subwindows as the rejection condition while the SegKHT aims at computing transformation on sliding windows efficiently; 3) as will be shown in the experimental results, the pattern matching using SegKHT is faster than IDA.

4.4.3 Relationship Between GCK and SegGCK

When the KHT matrix of the SegKHT matrix in (4.38) is a GCK matrix, we call this SegKHT the Segmented GCK (SegGCK). Note that the GCK matrix is in the form of $\mathbf{V}^{(N)} = \mathbf{M}^{(N_\mathbf{M})} \otimes \mathbf{S}_r$, where $\mathbf{M}^{(N_\mathbf{M})}$ is the $N_\mathbf{M} \times N_\mathbf{M}$ WHT matrix and $\mathbf{S}_r$ is a $R \times R$ matrix. Let % be the modulo operation. Denote the space spanned by the u basis vectors in matrix $\mathbf{V}^{(u\times N)} = [\vec{\mathbf{m}}^{(N,0)} \ \ldots \vec{\mathbf{m}}^{(N,u-1)}]^T$ by $span(\mathbf{V}^{(u\times N)})$. The following theorem describes the links between GCK and SegGCK:

Theorem 4.4 If L_s GCK basis vectors in $\mathbf{V}_{GCK}^{(L_s\times N)}$ can be represented as:

$$\vec{\mathbf{v}}^{(N,i)} = \vec{\mathbf{m}}^{(N_\mathbf{M},kL_s+i_s)} \otimes \vec{\mathbf{s}}_r^{(i_r)}, \text{for } i_s = 0,\ldots,L_s-1, \tag{4.46}$$

where $\vec{\mathbf{s}}_r^{(i_r)}$ is fixed, $N = N_\mathbf{M}R$, $N_\mathbf{M}\%L_s = 0, k = 0,\ldots,\frac{N_\mathbf{M}}{L_s}-1$ and WHT vectors $\vec{\mathbf{m}}^{(N_\mathbf{M},\cdot)}$ are in dyadic order or sequency order, then we can find L_s SegGCK basis vectors in $\mathbf{V}_{SegGCK}^{(L_s\times N)}$ so that:

1. $span(\mathbf{V}_{SegGCK}^{(L_s\times N)}) = span(\mathbf{V}_{GCK}^{(L_s\times N)})$.
2. $\forall\vec{\mathbf{x}}$, $||\mathbf{V}_{GCK}^{(L_s\times N)}\vec{\mathbf{x}}||^2 = ||\mathbf{V}_{SegGCK}^{(L_s\times N)}\vec{\mathbf{x}}||^2$.
3. $4/(3L_s)$ additions and $4/3$ additions per basis vector per window are required for computing $\mathbf{V}_{SegGCK}^{(L_s\times N)}\vec{\mathbf{x}}$ and $\mathbf{V}_{GCK}^{(L_s\times N)}\vec{\mathbf{x}}$ respectively, thus the computation required for SegGCK is $1/L_s$ of that required for GCK.

The proof for the theorem above is provided in Appendix A. According to the $||\mathbf{V}_{GCK}^{(L_s\times N)}\vec{\mathbf{x}}||^2 = ||\mathbf{V}_{SegGCK}^{(L_s\times N)}\vec{\mathbf{x}}||^2$ in Theorem 4.4, u GCK and SegGCK basis vectors pack the same energy and reject the same mismatched candidates when $u = L_s, 2L_s, 3L_s, \ldots$ for transform domain pattern matching in Table 1.1. Theorem 4.4 can be used for WHT and SegWHT when we set $\vec{\mathbf{s}}_r^{(i_r)} = \mathbf{I}_1$ in (4.46). We use the example in (4.41) for illustrating the relationship between GCK and SegGCK. Fig. 4.8 shows that all order-8 WHT basis vectors can be linearly represented by SegWHT basis vectors. For example,

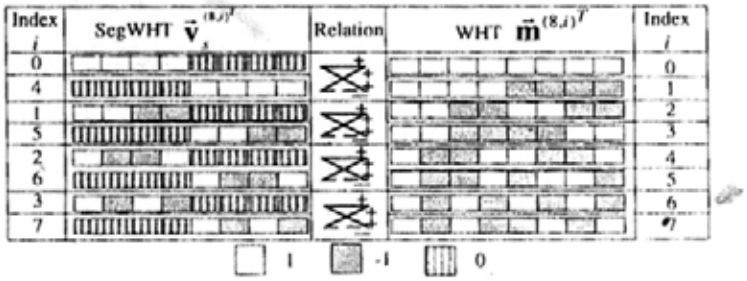


Figure 4.8: *The linear relationship among order-8 SegWHT and order-8 WHT, e.g.* $\vec{\mathbf{m}}^{(8,0)} = \vec{\mathbf{v}}_s^{(8,0)} + \vec{\mathbf{v}}_s^{(8,4)}$, $\vec{\mathbf{m}}^{(8,1)} = \vec{\mathbf{v}}_s^{(8,0)} - \vec{\mathbf{v}}_s^{(8,4)}$. *White represents the value +1, grey represents the value −1 and vertical strips represent the value 0. Normalization factors of basis vectors are skipped.*

when $k = 0$, we have the $L_s = 2$ dyadic ordered WHT vectors $\vec{\mathbf{m}}^{(8,0)}$ and $\vec{\mathbf{m}}^{(8,1)}$, where $\vec{\mathbf{s}}_r^{(i_r)} = \mathbf{I}_1, i_{\mathbf{M}} = 0, 1, L_s = 2, R = 1, N = N_{\mathbf{M}} = 8$ and $N_{\mathbf{M}} \% L_s = 8 \% 2 = 0$ in (4.46). And we select $L_s (= 2)$ SegWHT vectors $\vec{\mathbf{v}}_s^{(8,0)}$ and $\vec{\mathbf{v}}_s^{(8,4)}$ in (4.41). Orthonormal WHT basis vectors $\vec{\mathbf{m}}^{(8,0)}$ and $\vec{\mathbf{m}}^{(8,1)}$ can be linearly represented by orthonormal SegWHT basis vectors $\vec{\mathbf{v}}_s^{(8,0)}$ and $\vec{\mathbf{v}}_s^{(8,4)}$. Thus $span([\vec{\mathbf{m}}^{(8,0)}\ \vec{\mathbf{m}}^{(8,4)}]^T) = span([\vec{\mathbf{v}}_s^{(8,0)}\ \vec{\mathbf{v}}_s^{(8,4)}]^T)$ and $\forall \vec{\mathbf{x}}$, $|||[\vec{\mathbf{m}}^{(8,0)}\ \vec{\mathbf{m}}^{(8,4)}]^T \vec{\mathbf{x}}||^2 = |||[\vec{\mathbf{v}}_s^{(8,0)}\ \vec{\mathbf{v}}_s^{(8,4)}]^T \vec{\mathbf{x}}||^2$. As for computation, the fast SegKHT algorithm computes the SegWHT basis vectors by about 2/3 additions per basis vector per window as we have illustrated for (4.44) and (4.45). The KHT algorithm computes the WHT basis vectors by 4/3 additions per window. Thus the computation required for SegWHT is $1/L_s (= 1/2)$ of that required for WHT.

4.5 Advantage of KHT

We shall analyze the advantage of KHT by illustrating that the application of KHT in the transform domain pattern matching can achieve better performance compared with WHT, the GCK and the generalized GCK.

As introduced in Section 1.2.4, transform domain pattern matching requires that the transformation should be computationally efficient in packing energy.

As a comparison of the generalized GCK and the GCK, the generalized GCK requires more operations for obtaining projection values than the GCK. In the Appendix B, we prove that if the generalized GCK matrix is not a GCK matrix, then the basis vectors in the generalized GCK matrix are not orthogonal to each other. As mentioned in [32], nonorthogonal transform requires significantly more computation for evaluating the rejection condition. Thus the GCK seems to be the right choice from the generalized GCK for transform domain pattern matching. Compared with the GCK, the

KHT provides more choices of orthogonal transforms that can be computed efficiently.

Among the families of the GCK and the generalized GCK, the most efficient transform domain pattern matching reported uses WHT because: 1) WHT has good energy packing ability; 2) the computation of WHT requires only 2 additions per projection value using the GCK algorithm.

In the followings of this section, we find two subsets of KHT that: 1) cannot be represented by GCK; 2) can be more computationally efficient in packing energy and so is more efficient in pattern matching than WHT.

WHT requires that the size of input data N should be power of 2. To compare with WHT, we restrict N to be power of 2 in the following examples.

4.5.1 Transform Coding Gain on Statistical Model

Transform coding gain on statistical model is a commonly used criterion for measuring energy packing ability [63; 64]. Let the vector $\vec{\mathbf{x}}^{(N)}$ be a one dimensional, zero-mean, unit-variance, first-order Markov process [65] with adjacent element correlation ρ and covariance matrix $\mathbf{Cov}(\vec{\mathbf{x}}^{(N)})$, where

$$\begin{aligned}\mathbf{Cov}(\vec{\mathbf{x}}^{(N)}) &= E[\vec{\mathbf{x}}^{(N)} \cdot \vec{\mathbf{x}}^{(N)^T}] \\ &= \begin{bmatrix} 1 & \rho & \rho^2 & \dots & \rho^{N-1} \\ \rho & 1 & \rho & \dots & \rho^{N-2} \\ \rho^2 & \rho & 1 & \dots & \rho^{N-3} \\ \dots & \dots & \dots & \dots & \dots \\ \rho^{N-1} & \rho^{N-2} & \rho^{N-3} & \dots & 1 \end{bmatrix}\end{aligned} \tag{4.47}$$

and $E[\cdot]$ denotes expected value. The coding gain of a transform $\mathbf{V}^{(N)}$ is defined on the covariance matrix $\mathbf{Cov}(\vec{\mathbf{y}}^{(N)})$, where

$$\begin{aligned}\vec{\mathbf{y}}^{(N)} &= \mathbf{V}^{(N)}\vec{\mathbf{x}}^{(N)}, \\ \mathbf{Cov}(\vec{\mathbf{y}}^{(N)}) &= E[\vec{\mathbf{y}}^{(N)} \cdot \vec{\mathbf{y}}^{(N)^T}] \\ &= \begin{bmatrix} R(\vec{\mathbf{y}}^{(N)})_{0,0} & \dots & R(\vec{\mathbf{y}}^{(N)})_{0,N-1} \\ \dots & \dots & \dots \\ R(\vec{\mathbf{y}}^{(N)})_{N-1,0} & \dots & R(\vec{\mathbf{y}}^{(N)})_{N-1,N-1} \end{bmatrix}.\end{aligned} \tag{4.48}$$

Coding gain G_{TC} is given by

$$G_{TC} = \frac{\sum_{n=0}^{N-1} R(\vec{\mathbf{y}}^{(N)})_{n,n}}{\left(\prod_{n=0}^{N-1} R(\vec{\mathbf{y}}^{(N)})_{n,n}\right)^{1/N}}, \tag{4.49}$$

where $R(\vec{y}^{(N)})_{n,n}$ represents the variance of the nth projection value. The larger is the coding gain, the greater is the energy packing ability of the transformation.

4.5.2 Example 1 of KHT - SegKHT

The first example we choose is the SegKHT which is the Kronecker product of identity matrix $\mathbf{I}_{L_s}$ and WHT matrix $\mathbf{M}^{(N_\mathbf{M})}$ as follows:

$$\mathbf{V}^{(N)} = \mathbf{I}_{L_s} \otimes \mathbf{M}^{(N_\mathbf{M})} \otimes \mathbf{I}_1, \text{where } N = L_s N_\mathbf{M}. \tag{4.50}$$

The SegKHT matrix above is a KHT matrix where $\mathbf{S}_l = \mathbf{I}_{L_s}$ and $\mathbf{S}_r = \mathbf{I}_1$ but is not a GCK matrix. This SegKHT is better than WHT because: 1) as analyzed in Section 4.4.2, computation of the SegKHT is more efficient than computation of WHT; 2) as will be analyzed in the followings of this section, the SegKHT in (4.50) is almost the same as WHT in energy packing ability. In Section 4.6, we use experimental results to show that this SegKHT is more efficient than WHT in pattern matching.

We use coding gain to measure the energy packing ability. The coding gain of the DCT, KHT and WHT for two different sizes as a function of correlation coefficients ρ are shown in Fig. 4.9. The sizes $N = 256$ and $N = 1024$ can be considered as 16×16 patterns and 32×32 patterns respectively. The DCT is used as a benchmark, which is known to have better energy packing ability than WHT. It can be seen from Fig. 4.9 that the coding gain of SegKHT in (4.50) and WHT are almost overlapping with each other. Thus the SegKHT in the form of KHT $\mathbf{V}_s^{(N)} = \mathbf{I}_8 \otimes \mathbf{M}^{(N_\mathbf{M})}$, which is not GCK, has almost the same energy packing ability as WHT. The KHT $\mathbf{V}^{(N)} = \mathbf{M}^{(N_\mathbf{M})} \otimes \mathbf{I}_8$ is not a good choice for pattern matching because its coding gain is obviously worse than both WHT and SegKHT. Although the DCT has better energy packing ability, it is less efficient than WHT in pattern matching because of its relatively high computational complexity.

4.5.3 Example 2 of KHT - Kronecker Product of Haar Transform and WHT

The second example of KHT is in the form of $\mathbf{V}^{(N)} = \mathbf{S}_l \otimes \mathbf{M}^{(N_\mathbf{M})} \otimes \mathbf{S}_r$, where $N = LN_\mathbf{M}$, $\mathbf{S}_l$ is a $L \times L$ two scale Haar wavelet transform matrix and $\mathbf{S}_r = \mathbf{I}_1$. We shall show that this KHT is also better than WHT. For example, the Kronecker product of two scale

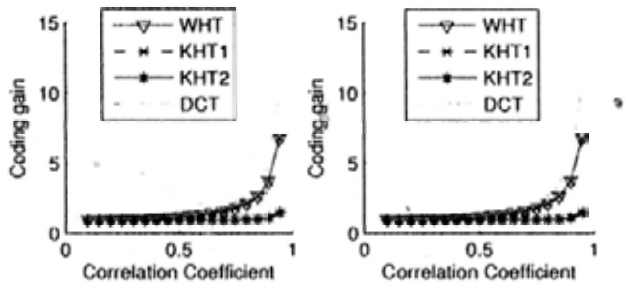


Figure 4.9: *The coding gain G_{TC} of the DCT, KHT and WHT on different correlation coefficients ρ ranging from 0.1 to 0.95. The left figure is for input window size $N = 256$ and the right figure is for $N = 1024$. KHT1 and KHT2 are in the form of $\mathbf{V}^{(N)} = \mathbf{I}_8 \otimes \mathbf{M}^{(N/8)}$ and $\mathbf{V}^{(N)} = \mathbf{M}^{(N/8)} \otimes \mathbf{I}_8$ respectively.*

Haar wavelet transform matrix of size 4×4 and order-2 WHT $\mathbf{M}^{(2)}$ is as follows:

$$\mathbf{V}^{(8\times 8)} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{4.51}$$

For the experimental results shown in Fig. 4.10 , we choose $\mathbf{V}^{(N)} = \mathbf{S}_l \otimes \mathbf{M}^{(8)} \otimes \mathbf{S}_r$, where $N = 256$ or 1024, $\mathbf{S}_l$ is an $N/8 \times N/8$ two scale Haar wavelet transform matrix and $\mathbf{S}_r = \mathbf{I}_1$. For this kind of KHT, basis vectors $\vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(N_M,0)}$ for $i_L = 0, \ldots, L-1$ will be Haar-like features and can be computed efficiently by integral image in [48]. Thus projection values corresponding to these basis vectors will act as the first projection value and the other projection values can be computed efficiently from the first projection value using the KHT algorithm.

The percentage of energy packed into the first u projection values for the statistical model introduced in Section 4.5.1 is measured by:

$$PER^{(u)} = \frac{\sum_{n=0}^{u-1} R(\vec{\mathbf{y}})_{n,n}}{\sum_{n=0}^{N-1} R(\vec{\mathbf{y}})_{n,n}}, \tag{4.52}$$

Analysis on this measure is presented by Kitajima [66] and by Yip and Rao [67]. In our experiment, we set correlation coefficient ρ as 0.9, which is a reasonable setting for images.

The percentage of energy extracted as a function of the number of operations per pixel required by different transforms is shown in Fig. 4.10 for input data having size $N = 256$. To evaluate this percentage as a function of the number of operations, we first

find the minimum number of projection values u required by different transformations for packing the same percentage of energy, and then evaluate the number of additions required for computing these u projection values for different transformations. It can be seen that this KHT can extract energy from input data using much fewer number of additions compared with WHT.

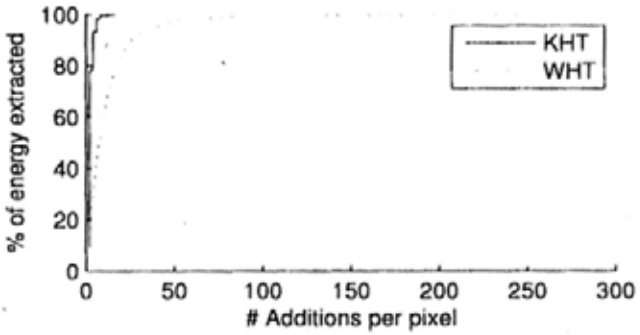


Figure 4.10: *The percentage of energy extracted as a function of the number of additions per pixel required by WHT and KHT for input data having size $N = 256$. In the experiment, we set $\rho = 0.9$ for Markov process input data in (4.47). KHT is in the form of $\mathbf{V}^{(N)} = \mathbf{S}_l \otimes \mathbf{M}^{(8)} \otimes \mathbf{S}_r$, where $\mathbf{S}_l$ is the two scale $N/8 \times N/8$ Haar wavelet transform matrix and $\mathbf{S}_r = \mathbf{I}_1$.*

4.6 Experimental Results

This section evaluates the performance of KHT algorithm and SegKHT by comparing them with FS and the other FS equivalent algorithms in pattern matching. All of the experiments were implemented on a 2.13GHz PC using C on windows XP system with compiling environment VC 6.0. Sum of squared difference (SSD) is used for measuring the dissimilarity between pattern and candidate windows.

4.6.1 Dataset and Algorithms Used for Pattern Matching Experiment

To investigate the computational efficiency of the proposed SegKHT for pattern matching, we shall compare the following FS equivalent algorithms with FS:

- 1) FFT: the FFT-based approach in OpenCV [53];
- 2) IDA: the IDA algorithm in [31];
- 3) WHT: the WHT algorithm for WHT in [1];
- 4) GCK: the GCK algorithm for WHT in [32];
- 5) WHT_{KHT}: the proposed KHT algorithm for WHT;

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

Table 4.4: *Datasets and corresponding sizes of images and patterns used in the experiments.*

- 6) SegKHT: the proposed KHT algorithm for SegKHT.

The code for WHT is available online [57] and the code for IDA is provided by the authors of [31]. The parameters for WHT use the default values in [57] and those for IDA are chosen according to [31]. For GCK, we choose the sequency-ordered WHT and the code is based on the code used for motion estimation in [2]. WHT_{KHT} is in the order introduced in Fig. 4.5.

As explained in [1], when the percentage of remaining candidate windows is smaller than certain number, denoted as ϵ, it is more efficient to use FS for finding the matched windows instead of using transformation. In the experiments, the default setting is $\epsilon = 0.02\%$ and $L_s = 8$ for SegKHT. According to the authors' source code for WHT in [57], we set $\epsilon = 2\%$ as default value for WHT and GCK. We will give variations of L_s and ϵ in Section 4.6.4.

Table 4.4 shows the 6 datasets used for evaluating the performance of the compared algorithms. The dataset includes different sizes of patterns and images. Our experiments include a total of 120 images chosen among three databases: MIT [54], medical [55], and remote sensing [56]. 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 contain radiographs and Landsat satellite images. The 120 images have 4 resolutions which are 160×120, 320×240, 640×480 and 1280×960. Each resolution has 30 images. The OpenCV function 'cvResize' with linear interpolation is used for producing the desired resolution of 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 4.4 were formed. Each dataset has 300 image-pattern pairs. Datasets $S1$ to $S4$ are the same as those in [31]. 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. Similar to the experiment in [31], the threshold T for $N_1 \times N_2$ pattern is set as:

$$T = 1.1 \cdot SSD_{min} + N_1 N_2, \tag{4.53}$$

where SSD_{min} is the SSD between the pattern and the best matching window.

Since all the evaluated algorithms find the same matching windows as the FS, the only concern is computational efficiency which is measured by execution time in the experiments. The time speed-up of algorithm A over algorithm B is measured by the execution time required by B divided by that required by A. As an example, the time speed-up of GCK over FS is measured as the execution time required by FS divided by that required by GCK. The larger is the speed-up, the faster is GCK compared with FS.

All necessary preprocessing like the time required for transformation on pattern has been included in all experiments. The fast KHT algorithm and the GCK algorithm require that one projection value on all window positions has been computed by other approaches. This projection value is the *dc* component in the experiment and computed using the box-technique [47] by 4 additions per window position. The computation required for *dc* component has been included in all experiments for the related algorithms.

4.6.2 Experiment 1 – Different Image-Pattern Sizes

In this experiment, we compare the time speed-ups yielded by the considered algorithms in pattern matching on the datasets $S1 - S4$ which have different sizes of image-pattern pairs. The time speed-ups yielded by GCK, IDA and SegKHT over the FS in pattern matching for each dataset are shown in Fig. 4.11. It can be seen that SegKHT outperforms the other compared algorithms in the four different datasets.

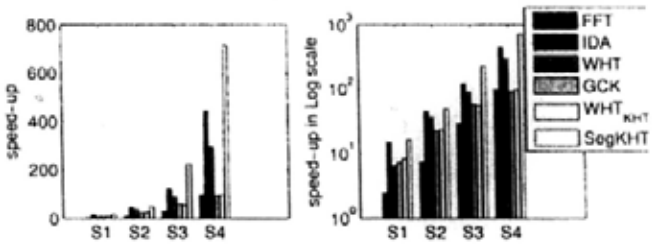


Figure 4.11: *Time speed-up over FS on datasets S1 – S4 for different algorithms measured by normal scale* (left) *and log scale* (right). *The bars for each dataset from left to right correspond to algorithms FFT, WHT, GCK, IDA, WHT_KHT and SegKHT.*

4.6.3 Experiment 2 – 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, we add 3 different kinds of noises having 4 noise levels to each image. The 4 Gaussian noises $G(1)$, $G(2)$, $G(3)$ and $G(4)$ range from low noise to high noise and have variances being respectively 100, 200, 400 and 800, where the 512×512 distorted "couple" images have PSNR 28.1, 25.1, 22.1 and 19.2 respectively when compared with the original image in Figure 1.1. The 4 different levels of Gaussian low pass filter are used for blurring each image, which are referred to as $B(1)$, $B(2)$, $B(3)$ and $B(4)$, correspond to Gaussian low pass filter having standard deviation σ=0.2, 0.9, 1.6 and 2.3, where the 512×512 distorted "couple" images have PSNR 27.79, 27.18, 25.36 and 24.14 respectively. The 4 different JPEG compression quality levels, which are referred to as $J(1)$, $J(2)$, $J(3)$ and $J(4)$, correspond to quality measure $Q_{JPG} = 90, 70, 50$ and 30 respectively, where higher Q_{JPG} means higher quality. They correspond to the 512×512 distorted "couple" images have PSNR 39.88, 34.93, 33.10 and 31.52 respectively.

The SSD_{min} in (4.53) is the SSD between the undistorted pattern and the distorted pattern. The distorted pattern and the distorted image have the same noise, e.g. $G(1)$. Since the threshold T in (4.53) is a bit greater than this SSD_{min}, the distorted pattern will always be found as a matched window if we put it into the image. Therefore, the miss rate is 0. The false positives using the threshold in (4.53) are not greater than 0.0025% in $S4$-$S6$ for the 4 noise levels in 3 different noise types. Fig. 4.13 shows the false positives for Gaussian noise in datasets $S4$-$S6$.

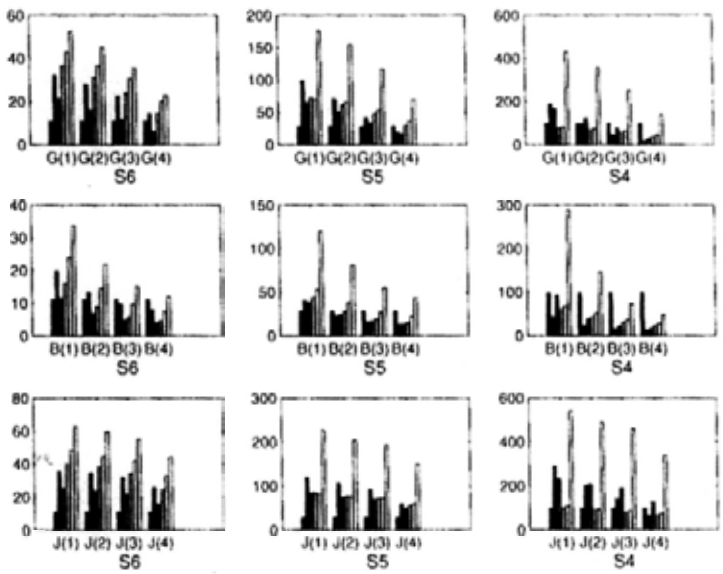


Figure 4.12: *Time speed-ups yielded by different algorithms over FS for Gaussian noise (upper row), image blur (middle row) and JPEG compression (bottom row) in different noise levels and different sizes of image-pattern pairs in pattern matching. Label of bars are the same as Fig. 4.11.*

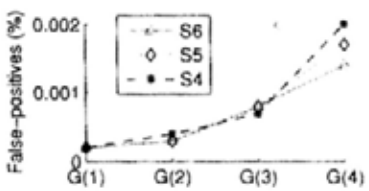


Figure 4.13: *False-positives (%) for noises $G(1)$-$G(4)$ in dataset $S4$-$S6$.*

Fig. 4.12 shows the speed-ups of examined fast algorithms over FS in different sizes of patterns and different levels of noises. It can be seen that SegKHT is the fastest and WHT transform using the KHT algorithm is faster than WHT transform using the GCK algorithm. The extensive experimental results of Tombari etc. in [31] have shown that FFT is faster than IDA and WHT for dataset $S4$ when the noise level is high. Our experimental results for dataset $S4$ also show that FFT is faster than WHT, GCK and IDA when noise level is high. However, SegKHT still outperforms FFT except for $B(3)$ and $B(4)$ in dataset $S4$.

4.6.4 Experiment 3 – Parameters

Experiments 1 and 2 use the default setting introduced in Section 4.6.1. This section investigates the influence of the parameters L_s and ϵ on the performance of pattern matching.

Fig. 4.14 shows the influence of L_s on time speed-up of SegKHT over FS in dataset $S4$. In this experiment, SegKHT under different settings of L_s always outperforms GCK in pattern matching. And we can find that the speed-up increases as L_s increases from 4 to 8 while speed-up decreases as L_s increases to 32 and 64. Both 8 and 16 are good choice of L_s. 8 is chosen as a default setting because the best setting of L_s is 8 in most cases although 16 is slightly better than 8 for noise level $G(1)$. Here is an analysis of the result in Fig. 4.14. The computational efficiency of transform domain pattern matching is determined by both the energy packing ability of transformation and the efficiency in computing transformation. Since the SegKHT requires $4/(3L_s)$ addition(s) per projection value, the larger is L_s, the more efficient is the transformation. However, the energy packing ability degrades as L_s increases. In the extreme, when $N_s = 1$, the SegKHT matrix is simply identity matrix $\mathbf{I}_{L_s}$. In this case, no computation is required for computing projection values, but the energy packing ability of SegKHT is the worst. Fig. 4.15(a) shows the number of remaining windows after each projection for GCK and SegKHT with different L_s on dataset $S4$ with Noise $G(4)$. It can be seen that as L_s increases, the rejection power of SegKHT decreases. For L_s being 4 and 16, the rejection power of SegKHT is close to WHT. As L_s increases to 64, the rejection power is seriously affected, which explains why $L_s = 64$ is less efficient than $L_s = 16$ in Fig. 4.14. Thus the decision of L_s is dependent on the trade-off between energy packing ability of projection values and the computation required for obtaining these projection values. As shown in Fig. 4.15(b), SegKHT using the SegKHT algorithm requires much fewer operations than WHT using the GCK algorithm in rejecting the same number of candidate windows. This results in better performance of SegKHT compared with GCK in Fig. 4.14.

As introduced in Section 4.4.3, it can be inferred from Theorem 4.4 that u 1D GCK and SegGCK basis vectors reject the same mismatched candidates when $u = L_s, 2L_s, 3L_s, \ldots$ for transform domain pattern matching in Table 1.1. This can be similarly used for 2D transforms. Thus we can see from Fig. 4.15(a) that SegKHT reject very similar amount of candidate windows at certain projection number, e.g. 16 for $L_s = 16$, 64 for $L_s = 64$.

As explained in [1], when the percentage of remaining candidate windows is smaller than certain number ϵ, it is more efficient to directly use SSD for finding the matched

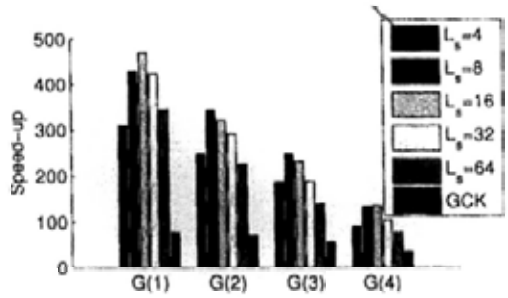


Figure 4.14: *The time speed-up over FS yielded by SegKHT with different L_s and GCK on dataset S4 with noise $G(1)$-$G(4)$.*

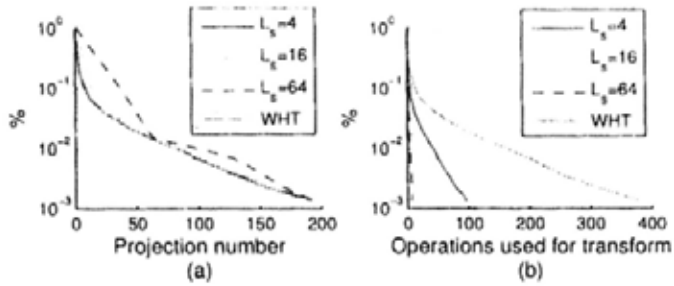


Figure 4.15: *The percentage of remaining windows as a function of the number of projections* (**a**) *and the number of operations required by transform* (**b**) *on dataset S4 with Noise $G(4)$.*

windows instead of using transformation. Fig. 4.16(a) shows the influence of ϵ for both GCK and SegKHT on dataset $S4$ with Gaussian noise $G(4)$. It can be seen that 2% is better for GCK while 0.02% is better for SegKHT. Fig. 4.16(b) shows that the computation of SegKHT is obviously faster than the computation of GCK for different situations of ϵ.

4.7 Summary

This chapter develops a family of transforms called Kronecker-Hadamard transform (KHT) that can be used for fast pattern matching using fast KHT algorithm. We then find that a subset of KHT, which is called Segmented KHT (SegKHT), can be

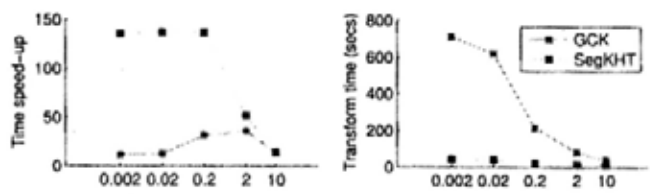


Figure 4.16: **(a)** *The speed-up over FS as a function of ϵ on the left figure and* **(b)** *the transformation time in seconds as a function of ϵ on the right figure. Experiments are done on dataset S4 with Noise $G(4)$. ϵ denotes the percentage of remaining window below which FS is used for pattern matching, e.g. 2 and 10 in the X axis correspond to 2% and 10% respectively.*

computed using a fast algorithm that is more efficient than the fast KHT algorithm.

The advantages of KHT and SegKHT are summarized as follows.

- KHT is a unified representation of Walsh Hadamard Transform (WHT) and Gray-Code Kernels (GCK).
- To achieve computational efficiency, a fast KHT algorithm is proposed. The KHT algorithm which requires 4/3 additions per datum per basis vector is faster than any known fast GCK algorithm.
- By segmenting input data into L_s parts, the fast computation of SegKHT requires $4/(3L_s)$ addition(s) per basis vector. SegKHT is a subset of KHT that cannot be represented by GCK.
- The computational cost of fast KHT and fast SegKHT algorithms are independent of the transform size and dimension.
- Basis vectors of KHT are orthogonal to each other, so are the basis vectors of SegKHT.

This chapter describes KHT and SegKHT in the context of transform domain pattern matching. However, pattern matching is only an example application. The properties of KHT and SegKHT make them attractive for many applications which require transformation on sliding windows such as image based rendering, image compression, super resolution, object detection, texture synthesis, block matching in motion estimation, image denoising, action recognition, wide baseline image matching, and more.

4.8 Appendix A: Proof for Theorem 4.4

This appendix provide proof for Theorem 4.4.

Proof: Restricted to the definition of WHT, L_s is a power of 2 value. We first

prove it when GCK is dyadic ordered WHT, then generalize it for GCK. The L_s dyadic-ordered WHT basis vectors $\vec{\mathbf{m}}^{(N,kL_s+i_s)}$ for $i_s = 0, 1, \ldots, L_s - 1$ can be represented by:

$$\begin{aligned}
&\begin{bmatrix} \vec{\mathbf{m}}^{(N,kL_s)^T} \\ \vec{\mathbf{m}}^{(N,kL_s+1)^T} \\ \vdots \\ \vec{\mathbf{m}}^{(N,kL_s+L_s-1)^T} \end{bmatrix} = \mathbf{M}^{(L_s)} \otimes \vec{\mathbf{m}}^{(N_s,k)^T} \\
&= (\mathbf{M}^{(L_s)}\mathbf{I}_{L_s}) \otimes (\mathbf{I}_1 \vec{\mathbf{m}}^{(N_s,k)^T}) \\
&= (\mathbf{M}^{(L_s)} \otimes \mathbf{I}_1)(\mathbf{I}_{L_s} \otimes \vec{\mathbf{m}}^{(N_s,k)^T}) \\
&= \mathbf{M}^{(L_s)}(\mathbf{I}_{L_s} \otimes \vec{\mathbf{m}}^{(N_s,k)^T}),
\end{aligned} \tag{4.54}$$

where $N = L_s N_s$, the L_s dyadic-ordered WHT basis vectors $\vec{\mathbf{m}}^{(N,kL_s+i_s)}$ for $i_s = 0, 1, \ldots, L_s - 1$ on the left-hand side of (4.54) is represented by linear combination of the L_s SegWHT basis vectors in $\mathbf{I}_{L_s} \otimes \vec{\mathbf{m}}^{(N_s,k)^T}$ on the right-hand side of (4.54) and the orthogonal matrix $\mathbf{M}^{(L_s)}$ describes this linear combination relationship. Hence, the L_s dyadic-ordered WHT basis vectors can be linearly represented by L_s order-N SegWHT basis vectors and vise versa. The proof for sequency-ordered WHT is similar.

Considering GCK, we have the follows from (4.54):

$$\begin{aligned}
\mathbf{V}_{GCK}^{(L_s \times N)} &= \begin{bmatrix} \vec{\mathbf{v}}^{(N,kL_s)^T} \\ \vec{\mathbf{v}}^{(N,kL_s+1)^T} \\ \vdots \\ \vec{\mathbf{v}}^{(N,kL_s+L_s-1)^T} \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{m}}^{(N_\mathbf{M},kL_s)^T} \\ \vec{\mathbf{m}}^{(N_\mathbf{M},kL_s+1)^T} \\ \vdots \\ \vec{\mathbf{m}}^{(N_\mathbf{M},kL_s+L_s-1)^T} \end{bmatrix} \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \\
&= [\mathbf{M}^{(L_s)}(\mathbf{I}_{L_s} \otimes \vec{\mathbf{m}}^{(N_\mathbf{M}/L_s,k)^T})] \otimes (\mathbf{I}_1 \vec{\mathbf{s}}_r^{(i_r)^T}) \\
&= \mathbf{M}^{(L_s)}(\mathbf{I}_{L_s} \otimes \vec{\mathbf{m}}^{(N_\mathbf{M}/L_s,k)^T} \otimes \vec{\mathbf{s}}_r^{(i_r)^T}) \\
&= \mathbf{M}^{(L_s)}(\mathbf{I}_{L_s} \otimes \vec{\mathbf{v}}^{(N_s,k)^T}) = \mathbf{M}^{(L_s)}\mathbf{V}_{SegGCK}^{(L_s \times N)},
\end{aligned} \tag{4.55}$$

where $\vec{\mathbf{s}}_r^{(i_r)}$ has length R, $N = N_\mathbf{M} R$, $N_s = N_\mathbf{M} R / L_s$, the L_s dyadic-ordered orthonormal GCK basis vectors in $\mathbf{V}_{GCK}^{(L_s \times N)}$ on the left-hand side of (4.55) are represented by linear combination of the L_s orthonormal SegGCK basis vectors in $\mathbf{V}_{SegGCK}^{(L_s \times N)}$ on the right-hand side of (4.55) and the orthogonal matrix $\mathbf{M}^{(L_s)}$ describes this linear combination relationship. Therefore, we have: $span(\mathbf{V}_{SegGCK}^{(L_s \times N)}) = span(\mathbf{V}_{GCK}^{(L_s \times N)})$; 2) $\forall \vec{\mathbf{x}}$, $\|\mathbf{V}_{GCK}^{(L_s \times N)} \vec{\mathbf{x}}\|^2 = \|\mathbf{V}_{SegGCK}^{(L_s \times N)} \vec{\mathbf{x}}\|^2$.

As for the computational complexity, SegGCK basis vectors can be computed by $4/(3L_s)$ additions per basis vector per window using the fast algorithm in Section 4.4.2 and the GCK basis vectors are computed by about 4/3 additions using the KHT algorithm.

4.9 Appendix B: Proof on the Generalized GCK

In this appendix, we prove that if the generalized GCK is not the GCK then the basis vectors of generalized GCK are not orthogonal to each other.

Proof: If the generalized GCK is not the GCK, then $\exists g, g \in 0, \ldots, G_{\mathrm{M}} - 1$, such that $a_g \neq \pm 1$. We prove that $\forall g, g \in 0, \ldots, G_{\mathrm{M}} - 1$, if $a_g \neq \pm 1$ then we can find two basis vectors that are not orthogonal to each other. Consider the following two generalized GCK basis vectors:

$$\begin{bmatrix} \vec{\mathbf{v}}^{(N,n_1)^T} \\ \vec{\mathbf{v}}^{(N,n_2)^T} \end{bmatrix} = \vec{\mathbf{s}}_{l_1}^{T} \otimes \begin{bmatrix} 1 & a_g \\ 1 & -a_g \end{bmatrix} \otimes \vec{\mathbf{s}}_{r_1}^{T}, \tag{4.56}$$

$$\text{where } \vec{\mathbf{s}}_{l_1}^{T} = \otimes_{k=0}^{g-1} [1 \quad \pm a_k], \tag{4.57}$$

$$\vec{\mathbf{s}}_{r_1}^{T} = (\otimes_{k=g+1}^{G_{\mathrm{M}}-1} [1 \quad \pm a_k]) \otimes \vec{\mathbf{s}}_{r}^{T}, \tag{4.58}$$

and $\vec{\mathbf{s}}_r$ is a basis vector in $\mathbf{S}_r$. We have follows for the two basis vectors in (4.56):

$$\begin{aligned} & \vec{\mathbf{v}}^{(N,n_1)^T} \vec{\mathbf{v}}^{(N,n_2)} \\ &= (\vec{\mathbf{s}}_{l_1} \otimes [1 \quad a_g]^T \otimes \vec{\mathbf{s}}_{r_1})^T (\vec{\mathbf{s}}_{l_1} \otimes [1 \quad -a_g]^T \otimes \vec{\mathbf{s}}_{r_1}) \\ &= (\vec{\mathbf{s}}_{l_1}^{T} \vec{\mathbf{s}}_{l_1}) \otimes ([1 \quad a_g][1 \quad -a_g]^T) \otimes (\vec{\mathbf{s}}_{r_1}^{T} \vec{\mathbf{s}}_{r_1}) \\ &= ||\vec{\mathbf{s}}_{l_1}||^2 ||\vec{\mathbf{s}}_{r_1}||^2 [1 - (a_g)^2]. \end{aligned} \tag{4.59}$$

Note that $||\vec{\mathbf{s}}_{l_1}||^2 = 0$ or $||\vec{\mathbf{s}}_{r_1}||^2 = 0$ in (4.59) corresponds to $\vec{\mathbf{v}}^{(N,n_1)} = \vec{\mathbf{v}}^{(N,n_2)} = 0$, which is not useful and thus not considered to happen for generalized GCK basis vectors. If $a_g \neq \pm 1$, then $[1 - (a_g)^2] \neq 0$ and $\vec{\mathbf{v}}^{(N,n_1)^T} \vec{\mathbf{v}}^{(N,n_2)} \neq 0$. Thus the two basis vector in (4.56) are not orthogonal to each other.

4.10 Appendix C: The KHT algorithm for D dimensional KHT

Consider D dimensional input window $\mathbf{X}^{(\vec{\mathbf{N}}\vec{\mathbf{j}})}$, where $\vec{\mathbf{N}} = [N_1 \ldots N_D]^T$ denotes the size and $\vec{\mathbf{j}} = [j_1, \ldots, j_D]^T$ denotes the position. Elements of $\mathbf{X}^{(\vec{\mathbf{N}}\vec{\mathbf{j}})}$ are denoted as $x_{j_1+n_1,\ldots,j_D+n_D}$, for $n_d = 0, 1, \ldots, N_d - 1$, $d = 1, \ldots, D$. To make it short, we denote

$\mathbf{X}^{(\vec{\mathbf{N}},\vec{\mathbf{j}})}$ at position $\vec{\mathbf{j}}$ as $\mathbf{X}_1$.

Four D dimensional projection values $y(\vec{\mathbf{N}},\vec{\mathbf{i}}^{(n)},\vec{\mathbf{j}})$ for $n = 0,1,2,3$ can be represented as follows:

$$y(\vec{\mathbf{N}},\vec{\mathbf{i}}^{(n)},\vec{\mathbf{j}}) = \sum_{k_1,\ldots,k_D} v_{k_1}^{(n)} \ldots v_{k_D}^{(n)} x_{j_1+k_1,\ldots,j_D+k_D}, \tag{4.60}$$

where $v_{k_d}^{(n)}$ for $d = 1,\ldots,D$ is the k_dth element of the KHT basis vector at dimension d. For example, 1D KHT can be represented as follows corresponding to (4.60):

$$y(N,i^{(n)},j) = \vec{\mathbf{v}}^{(N,i^{(n)})^T} \vec{\mathbf{x}}_w^{(N,j)} = \sum_{k=0}^{N-1} v_k^{(n)} x_{j+k}, \tag{4.61}$$

We define the α-index for D dimensional KHT as $[\alpha^{(1)} \ \cdots \alpha^{(d)} \ \cdots \alpha^{(D)}]$, where $\alpha^{(d)} = \alpha_{G_{\mathbf{M}_d}-1}^{(d)} \ldots \alpha_{g_d}^{(d)} \ldots \alpha_0^{(d)}$ is the α-index for the basis vector in dimension d, $N_{\mathbf{M}_d} = 2^{G_{\mathbf{M}_d}}$ and $\alpha_{g_d}^{(d)} = \{0,1\}$ for $g_d = 0,\ldots,G_{\mathbf{M}_d}-1$. Four D dimensional projection values in (4.60) are α^2-related at g_d if these four projection values have the same KHT basis vector on all dimensions except dimension d and the α-indices of their basis vectors in dimension d are only different at $\alpha_{g_d}^{(d)}$ and $\alpha_{g_d+1}^{(d)}$. Thus the four projection values represented in (4.60) have the same $v_{k_1}^{(n)} \ldots v_{k_D}^{(n)}$ for $n = 0,1,2,3$ except that they have different $v_{k_d}^{(n)}$. And we can consider the operation related to $v_{k_1}^{(n)},\ldots,v_{k_D}^{(n)}$ without k_d in (4.60) as a linear function f and have:

$$y(\vec{\mathbf{N}},\vec{\mathbf{i}}^{(n)},\vec{\mathbf{j}}) = f(\sum_{k_d} v_{k_d}^{(n)} x_{j_1+k_1,\ldots,j_D+k_D}), \tag{4.62}$$

Denote the input data sliding $R_{4_d} = 2^{G_{M_d}-g_d-2R_d}$ positions at dimension d from current position $\vec{\mathbf{j}}$ as $\mathbf{X}_2$, where $\mathbf{X}_2 = \mathbf{X}^{(\vec{\mathbf{N}},\vec{\mathbf{j}}_2)}$, $\vec{\mathbf{j}}_2 = [j_1,\ldots,j_d+R_{4_d},\ldots,j_D]^T$. We have

$$y(\vec{\mathbf{N}},\vec{\mathbf{i}}^{(n)},\vec{\mathbf{j}}_2) = f(\sum_{k_d} v_{k_d}^{(n)} x_{j_1+k_1,\ldots,j_d+R_{4_d}+k_d,\ldots,j_D+k_D}), \tag{4.63}$$

Using Corollary 3, we have:

$$\begin{aligned}
&\sum_{k_d} v_{k_d}^{(0)} x_{\ldots,j_d+k_d,\ldots} - \sum_{k_d} v_{k_d}^{(0)} x_{\ldots,j_d+R_{4_d}+k_d,\ldots} \\
&= \sum_{k_d} v_{k_d}^{(3)} x_{\ldots,j_d+k_d,\ldots} + \sum_{k_d} v_{k_d}^{(1)} x_{\ldots,j_d+R_{4_d}+k_d,\ldots} \\
&= \sum_{k_d} v_{k_d}^{(2)} x_{\ldots,j_d+k_d,\ldots} + \sum_{k_d} v_{k_d}^{(2)} x_{\ldots,j_d+R_{4_d}+k_d,\ldots} \\
&= \sum_{k_d} v_{k_d}^{(1)} x_{\ldots,j_d+k_d,\ldots} - \sum_{k_d} v_{k_d}^{(3)} x_{\ldots,j_d+R_{4_d}+k_d,\ldots} .
\end{aligned} \tag{4.64}$$

Since the f in (4.62) is a linear function, we have the follows from (4.62) by putting f for each term in (4.64):

$$\begin{aligned}
&y(\vec{\mathbf{N}}, \vec{\mathbf{i}}^{(0)}, \vec{\mathbf{j}}) - y(\vec{\mathbf{N}}, \vec{\mathbf{i}}^{(0)}, \vec{\mathbf{j}}_2) = y(\vec{\mathbf{N}}, \vec{\mathbf{i}}^{(3)}, \vec{\mathbf{j}}) + y(\vec{\mathbf{N}}, \vec{\mathbf{i}}^{(1)}, \vec{\mathbf{j}}_2) \\
&= y(\vec{\mathbf{N}}, \vec{\mathbf{i}}^{(2)}, \vec{\mathbf{j}}) + y(\vec{\mathbf{N}}, \vec{\mathbf{i}}^{(2)}, \vec{\mathbf{j}}_2) = y(\vec{\mathbf{N}}, \vec{\mathbf{i}}^{(1)}, \vec{\mathbf{j}}) - y(\vec{\mathbf{N}}, \vec{\mathbf{i}}^{(3)}, \vec{\mathbf{j}}_2).
\end{aligned} \tag{4.65}$$

The relationship in (4.65) for D dimensional KHT corresponds to the relationship for 1D KHT in Corollary 3. Thus the KHT algorithm for D dimensional separable KHT is independent of dimension and size.

4.11 Appendix D: Proof of Theorem 4.2

Theorem 4.2 If four order-N KHT projection values $y(N, i^{(n)}, j) = [\vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(N_{\mathbf{M}}, i_{\mathbf{M}}^{(n)})} \otimes \vec{\mathbf{s}}_r^{(i_r)}]^T \vec{\mathbf{x}}_w^{(N,j)}$ for $n = 0, \ldots, 3$ as defined in (11) are α^2-related at g, where $i_{\mathbf{M}}^{(n)}$ are represented in (20)-(23), $\vec{\mathbf{s}}_l^{(i_l)}$ has length L and $\vec{\mathbf{s}}_r^{(i_r)}$ has length R, then we have:

$$\begin{bmatrix} y(N, i^{(1)}, j + R_4) \\ y(N, i^{(2)}, j + R_4) \\ y(N, i^{(3)}, j + R_4) \end{bmatrix} = \begin{bmatrix} -y(N, i^{(3)}, j) \\ -y(N, i^{(2)}, j) \\ y(N, i^{(1)}, j) \end{bmatrix} + \begin{bmatrix} \Delta(N, j, R_4) \\ \Delta(N, j, R_4) \\ -\Delta(N, j, R_4) \end{bmatrix}, \tag{4.66}$$

$$\text{where} \quad R_4 = 2^{G_{\mathbf{M}} - g - 2} R, \tag{4.67}$$

$$\Delta(N, j, R_4) = y(N, i^{(0)}, j) - y(N, i^{(0)}, j + R_4). \tag{4.68}$$

Proof:

We first prove the computation in (4.66)-(4.68) when KHT projection values have the form $y(N, i^{(n)}, j) = [\vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(4, i_{\mathbf{M}}^{(n)})} \otimes \vec{\mathbf{s}}_r^{(i_r)}]^T \vec{\mathbf{x}}_w^{(N,j)}$, where $\vec{\mathbf{s}}_l^{(i_l)}$ has length L_4, $\vec{\mathbf{m}}^{(4, i_{\mathbf{M}}^{(n)})}$ has length 4, $\vec{\mathbf{s}}_r^{(i_r)}$ has length R_4, $N = 4L_4R_4$. In this form, we have $g = 0$, $N_{\mathbf{M}} = 4$

for WHT basis vector $\vec{\mathbf{m}}^{(N_M, i_M^{(n)})}$ and:

$$\begin{bmatrix} \vec{\mathbf{v}}_N(i^0)^T \\ \vec{\mathbf{v}}_N(i^1)^T \\ \vec{\mathbf{v}}_N(i^2)^T \\ \vec{\mathbf{v}}_N(i^3)^T \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{s}}_l^{(i_l)^T} \otimes \vec{\mathbf{m}}^{(4,0)^T} \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \\ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes \vec{\mathbf{m}}^{(4,1)^T} \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \\ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes \vec{\mathbf{m}}^{(4,2)^T} \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \\ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes \vec{\mathbf{m}}^{(4,3)^T} \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \end{bmatrix} = \vec{\mathbf{s}}_l^{(i_l)^T} \otimes \mathbf{M}^{(4)} \otimes \vec{\mathbf{s}}_r^{(i_r)^T}. \tag{4.69}$$

According to the definition of $\vec{\mathbf{v}}^{(N,i)}$ and $y(N,i,j)$ we have:

$$\begin{aligned} y(N, i^{(0)}, j) &= \vec{\mathbf{v}}^{(N,i^{(0)})^T} \vec{\mathbf{x}}_w^{(N,j)} = \left\{ \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}_4(0) \otimes \vec{\mathbf{s}}_r^{(i_r)} \right\}^T \vec{\mathbf{x}}_w^{(N,j)} \\ &= \left\{ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1\ 1\ 1] \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \right\} \vec{\mathbf{x}}_w^{(N,j)} \\ &= \left\{ \left[\vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1\ 1\ 1] \quad 0 \right] \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \right\} \vec{\mathbf{x}}_w^{(N+R_4,j)}, \\ y(N, i^{(0)}, j + R_4) &= \left\{ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1\ 1\ 1] \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \right\} \vec{\mathbf{x}}_N(j + R_4) \\ &= \left\{ \left[0 \quad \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1\ 1\ 1] \right] \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \right\} \vec{\mathbf{x}}_w^{(N+R_4,j)}. \end{aligned} \tag{4.70}$$

Thus we have:

$$\begin{aligned} \Delta(N, j, R_4) &= y(N, i^{(0)}, j) - y(N, i^{(0)}, j + R_4) \\ &= \left\{ \left(\left[\vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1\ 1\ 1] \quad 0 \right] - \left[0 \quad \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1\ 1\ 1] \right] \right) \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \right\} \vec{\mathbf{x}}_w^{(N+R_4,j)}, \end{aligned} \tag{4.71}$$

Similarly:

$$\begin{aligned} &y(N, i^{(2)}, j) + y(N, i^{(2)}, j + R_4) \\ &= \left\{ \left(\left[\vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ -1\ 1\ -1] \quad 0 \right] + \left[0 \quad \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ -1\ 1\ -1] \right] \right) \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \right\} \vec{\mathbf{x}}_w^{(N+R_4,j)}, \end{aligned} \tag{4.72}$$

$$\begin{aligned} &y(N, i^{(1)}, j) - y(N, i^{(3)}, j + R_4) \\ &= \left\{ \left(\left[\vec{\mathbf{s}}_l^{(i_l)^T} \otimes [\ 1\ \ 1\ \ -1\ \ -1\ ] \quad 0 \right] - \left[0 \quad \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [\ 1\ \ -1\ \ -1\ \ 1\ ] \right] \right) \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \right\} \vec{\mathbf{x}}_w^{(N+R_4,j)}, \end{aligned} \tag{4.73}$$

$$\begin{aligned} &y(N, i^{(3)}, j) + y(N, i^{(1)}, j + R_4) \\ &= \left\{ \left(\left[\vec{\mathbf{s}}_l^{(i_l)^T} \otimes [\ 1\ \ -1\ \ -1\ \ 1\ ] \quad 0 \right] + \left[0 \quad \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [\ 1\ \ 1\ \ -1\ \ -1\ ] \right] \right) \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \right\} \vec{\mathbf{x}}_w^{(N+R_4,j)}. \end{aligned} \tag{4.74}$$

If the basis vectors in (4.71)-(4.74) are equal, then we have (4.66)-(4.68) when KHT projection values have the form $y(N, i^{(n)}, j) = [\vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(4, i_M^{(n)})} \otimes \vec{\mathbf{s}}_r^{(i_r)}]^T \vec{\mathbf{x}}_w^{(N,j)}$. The

followings prove that the basis vectors in (4.71)-(4.74) are equal. The basis vectors in (4.71)-(4.74) share the right Kronecker product of $\vec{\mathbf{s}}_r^{(i_r)^T}$, i.e. $\otimes\vec{\mathbf{s}}_r^{(i_r)^T}$, this part is skipped.

Let $\vec{\mathbf{s}}_l^{(i_l)}$ be represented as $[s_0\ s_1\ s_2\ \cdots, s_{L_4-1}]^T$. For (4.71), we have:

$$\begin{aligned}
&\left[\ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1\ 1\ 1]\ \ 0\ \right] - \left[\ 0\ \ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1\ 1\ 1]\ \right] \\
&= [s_0,\ s_0,\ s_0,\ s_0,\ s_1,\ s_1,\ s_1,\ s_1,\ \cdots, s_{L_4-1},\ s_{L_4-1},\ 0] \\
&\quad - [0,\ \ s_0,\ s_0,\ s_0,\ s_0,\ s_1,\ s_1,\ s_1,\ s_1,\ \cdots, s_{L_4-1},\ s_{L_4-1}] \\
&= [s_0,\ 0,\ 0,\ 0,\ s_1 - s_0,\ 0,\ 0,\ 0,\ s_2 - s_1,\ \cdots, 0, -s_{L_4-1}].
\end{aligned} \tag{4.75}$$

For (4.72), we have:

$$\begin{aligned}
&\left[\ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [\ 1\ \ -1\ \ 1\ \ -1\ ]\ \ 0\ \right] + \left[\ 0\ \ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [\ 1\ \ -1\ \ 1\ \ -1\ ]\ \right] \\
&= [s_0,\ -s_0,\ s_0,\ -s_0,\ s_1,\ -s_1,\ s_1,\ -s_1,\ \cdots, s_{L_4-1},\ -s_{L_4-1},\ 0] \\
&\quad + [0,\ \ s_0,\ -s_0,\ s_0,\ -s_0,\ s_1,\ -s_1,\ s_1,\ -s_1,\ \cdots, s_{L_4-1},\ -s_{L_4-1}] \\
&= [s_0,\ 0,\ 0,\ 0,\ s_1 - s_0,\ 0,\ 0,\ 0,\ s_2 - s_1,\ \cdots, 0, -s_{L_4-1}] \\
&= \left[\ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1\ 1\ 1]\ \ 0\ \right] - \left[\ 0\ \ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1\ 1\ 1]\ \right].
\end{aligned} \tag{4.76}$$

For (4.73), we have:

$$\begin{aligned}
&\left[\ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [\ 1\ \ 1\ \ -1\ \ -1\ ]\ \ 0\ \right] - \left[\ 0\ \ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [\ 1\ \ -1\ \ -1\ \ 1\ ]\ \right] \\
&= [s_0,\ s_0,\ -s_0,\ -s_0,\ s_1,\ s_1,\ -s_1,\ -s_1,\ \cdots, -s_{L_4-1},\ -s_{L_4-1},\ 0] \\
&\quad - [0,\ \ s_0,\ -s_0,\ -s_0,\ s_0,\ s_1,\ -s_1,\ -s_1,\ s_1,\ \cdots, -s_{L_4-1},\ s_{L_4-1}] \\
&= [s_0,\ 0,\ 0,\ 0,\ s_1 - s_0,\ 0,\ 0,\ 0,\ s_2 - s_1,\ \cdots, 0, -s_{L_4-1}] \\
&= \left[\ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1\ 1\ 1]\ \ 0\ \right] - \left[\ 0\ \ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1\ 1\ 1]\ \right].
\end{aligned} \tag{4.77}$$

For (4.74), we have:

$$\begin{aligned}
&\left[\ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [\ 1\ \ -1\ \ -1\ \ 1\ ]\ \ 0\ \right] + \left[\ 0\ \ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [\ 1\ \ 1\ \ -1\ \ -1\ ]\ \right] \\
&= [s_0,\ -s_0,\ -s_0,\ s_0,\ s_1,\ -s_1,\ -s_1,\ s_1,\ \cdots, -s_{L_4-1},\ s_{L_4-1},\ 0] \\
&\quad + [0,\ \ s_0,\ s_0,\ -s_0,\ -s_0,\ s_1,\ s_1,\ -s_1,\ -s_1,\ \cdots, -s_{L_4-1},\ -s_{L_4-1}] \\
&= [s_0,\ 0,\ 0,\ 0,\ s_1 - s_0,\ 0,\ 0,\ 0,\ s_2 - s_1,\ \cdots, 0, -s_{L_4-1}] \\
&= \left[\ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1\ 1\ 1]\ \ 0\ \right] + \left[\ 0\ \ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1\ 1\ 1]\ \right].
\end{aligned} \tag{4.78}$$

Thus, we have the followings when KHT projection values have the form $y(N, i^{(n)}, j) = [\vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(4, i_{\mathrm{M}}^{(n)})} \otimes \vec{\mathbf{s}}_r^{(i_r)}]^T \vec{\mathbf{x}}_w^{(N,j)}$:

$$\begin{aligned}\Delta(N, j, R_4) &= y(N, i^{(0)}, j) - y(N, i^{(0)}, j + R_4), \\ &= y(N, i^{(3)}, j) + y(N, i^{(1)}, j + R_4), \\ &= y(N, i^{(2)}, j) + y(N, i^{(2)}, j + R_4), \\ &= y(N, i^{(1)}, j) - y(N, i^{(3)}, j + R_4).\end{aligned} \tag{4.79}$$

Considering the general case when KHT has the form of $y(N, i^{(n)}, j) = [\vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(N_{\mathrm{M}}, i_{\mathrm{M}}^{(n)})} \otimes \vec{\mathbf{s}}_r^{(i_r)}]^T \vec{\mathbf{x}}_w^{(N,j)}$, we have the followings:

$$\begin{aligned}&\vec{\mathbf{v}}^{(N, i^{(0)})} = \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(N_{\mathrm{M}}, i_{\mathrm{M}}^{(0)})} \otimes \vec{\mathbf{s}}_r^{(i_r)} \\ &= \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(2,\alpha_0)} \otimes \cdots \otimes \vec{\mathbf{m}}^{(2,\alpha_{g-1})} \otimes \vec{\mathbf{m}}^{(2,0)} \otimes \vec{\mathbf{m}}^{(2,0)} \otimes \vec{\mathbf{m}}^{(2,\alpha_{g+2})} \otimes \cdots \otimes \vec{\mathbf{m}}^{(2,\alpha_{G_{\mathrm{M}}-1})} \otimes \vec{\mathbf{s}}_r^{(i_r)}, \\ &\vec{\mathbf{v}}^{(N, i^{(1)})} = \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(N_{\mathrm{M}}, i_{\mathrm{M}}^{(1)})} \otimes \vec{\mathbf{s}}_r^{(i_r)} \\ &= \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(2,\alpha_0)} \otimes \cdots \otimes \vec{\mathbf{m}}^{(2,\alpha_{g-1})} \otimes \vec{\mathbf{m}}^{(2,1)} \otimes \vec{\mathbf{m}}^{(2,0)} \otimes \vec{\mathbf{m}}^{(2,\alpha_{g+2})} \otimes \cdots \otimes \vec{\mathbf{m}}^{(2,\alpha_{G_{\mathrm{M}}-1})} \otimes \vec{\mathbf{s}}_r^{(i_r)}, \\ &\vec{\mathbf{v}}_N(i^{(2)}) = \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(N_{\mathrm{M}}, i_{\mathrm{M}}^{(2)})} \otimes \vec{\mathbf{s}}_r^{(i_r)} \\ &= \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(2,\alpha_0)} \otimes \cdots \otimes \vec{\mathbf{m}}^{(2,\alpha_{g-1})} \otimes \vec{\mathbf{m}}^{(2,0)} \otimes \vec{\mathbf{m}}^{(2,1)} \otimes \vec{\mathbf{m}}^{(2,\alpha_{g+2})} \otimes \cdots \otimes \vec{\mathbf{m}}^{(2,\alpha_{G_{\mathrm{M}}-1})} \otimes \vec{\mathbf{s}}_r^{(i_r)}, \\ &\vec{\mathbf{v}}_N(i^{(3)}) = \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(N_{\mathrm{M}}, i_{\mathrm{M}}^{(3)})} \otimes \vec{\mathbf{s}}_r^{(i_r)} \\ &= \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(2,\alpha_0)} \otimes \cdots \otimes \vec{\mathbf{m}}^{(2,\alpha_{g-1})} \otimes \vec{\mathbf{m}}^{(2,1)} \otimes \vec{\mathbf{m}}^{(2,1)} \otimes \vec{\mathbf{m}}^{(2,\alpha_{g+2})} \otimes \cdots \otimes \vec{\mathbf{m}}^{(2,\alpha_{G_{\mathrm{M}}-1})} \otimes \vec{\mathbf{s}}_r^{(i_r)}.\end{aligned} \tag{4.80}$$

Denote vector $\vec{\mathbf{s}}_{l_2}$ having length $2^g L$ and vector $\vec{\mathbf{s}}_{r_4}$ having length $2^{G_{\mathrm{M}}-g-2} R$ as follows:

$$\begin{aligned}\vec{\mathbf{s}}_{l_2} &= \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(2,\alpha_0)} \otimes \cdots \otimes \vec{\mathbf{m}}^{(2,\alpha_{g-1})}, \\ \vec{\mathbf{s}}_{r_4} &= \vec{\mathbf{m}}^{(2,\alpha_{g+2})} \otimes \cdots \otimes \vec{\mathbf{m}}^{(2,\alpha_{G_{\mathrm{M}}-1})} \otimes \vec{\mathbf{s}}_r^{(i_r)}.\end{aligned} \tag{4.81}$$

Then we have:

$$\begin{bmatrix} \vec{\mathbf{v}}^{(N, i^{(0)})^T} \\ \vec{\mathbf{v}}^{(N, i^{(1)})^T} \\ \vec{\mathbf{v}}_N(i^{(2)})^T \\ \vec{\mathbf{v}}_N(i^{(3)})^T \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{s}}_{l_2} \otimes \vec{\mathbf{m}}^{(4,0)^T} \otimes \vec{\mathbf{s}}_{r_4} \\ \vec{\mathbf{s}}_{l_2} \otimes \vec{\mathbf{m}}^{(4,1)^T} \otimes \vec{\mathbf{s}}_{r_4} \\ \vec{\mathbf{s}}_{l_2} \otimes \vec{\mathbf{m}}^{(4,2)^T} \otimes \vec{\mathbf{s}}_{r_4} \\ \vec{\mathbf{s}}_{l_2} \otimes \vec{\mathbf{m}}^{(4,3)^T} \otimes \vec{\mathbf{s}}_{r_4} \end{bmatrix} = \vec{\mathbf{s}}_{l_2} \otimes M_4 \otimes \vec{\mathbf{s}}_{r_4}. \tag{4.82}$$

Since $\vec{\mathbf{v}}^{(N, i^{(n)})}$ for $n = 0, 1, 2, 3$ in (4.82) are in the form shown in (4.69). We can set $L_4 = 2^g L$, $R_4 = 2^{G_{\mathrm{M}}-g-2} R$ in (4.69) and use the relationship in (4.79) to derive the

follows:

$$\begin{aligned}\Delta(N,j,R_4) &= y(N,i^{(0)},j) - y(N,i^{(0)},j+R_4),\\ &= y(N,i^{(3)},j) + y(N,i^{(1)},j+R_4),\\ &= y(N,i^{(2)},j) + y(N,i^{(2)},j+R_4),\\ &= y(N,i^{(1)},j) - y(N,i^{(3)},j+R_4).\end{aligned} \tag{4.83}$$

Then we have the Theorem 4.2 proved using (4.83) for deriving (4.66)-(4.68).

4.12 Appendix E: Proof that the GCK algorithm can be used for KHT

This appendix prove that the GCK algorithm can be used for KHT. In order to represent the GCK algorithm, we define the α-relation for 2 basis vectors and projection values. If the α-indices of $i_{\mathbf{M}}^{(0)}$ and $i_{\mathbf{M}}^{(1)}$ for $i_{\mathbf{M}}^{(0)}$, $i_{\mathbf{M}}^{(1)} = 0, 1, \ldots N_{\mathbf{M}} - 1$ are only different at α_g, then we say that: 1) WHT basis vectors $\vec{\mathbf{m}}^{(N_{\mathbf{M}},i_{\mathbf{M}}^{(0)})}$and $\vec{\mathbf{m}}^{(N_{\mathbf{M}},i_{\mathbf{M}}^{(1)})}$, are α-related at g; 2) the corresponding KHT projection values $y(N,i^{(0)},j)$ and $y(N,i^{(1)},j)$ having the same $\vec{\mathbf{s}}_l^{(i_l)}$ and the same $\vec{\mathbf{s}}_r^{(i_r)}$ are α-related at g, where $y(N,i^{(n)},j)$ for $n = 0, 1$ are represented as follows:

$$\begin{aligned}y(N,i^{(0)},j) &= [\vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(N_{\mathbf{M}},i_{\mathbf{M}}^{(0)})} \otimes \vec{\mathbf{s}}_r^{(i_r)}]^T \vec{\mathbf{x}}_w^{(N,j)},\\ y(N,i^{(1)},j) &= [\vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(N_{\mathbf{M}},i_{\mathbf{M}}^{(1)})} \otimes \vec{\mathbf{s}}_r^{(i_r)}]^T \vec{\mathbf{x}}_w^{(N,j)}.\end{aligned} \tag{4.84}$$

Without losing generality, we let the $i_{\mathbf{M}}^{(0)}$ and $i_{\mathbf{M}}^{(1)}$ in (4.84) be represented by:

$$i_{\mathbf{M}}^{(0)} = \alpha_{G_{\mathbf{M}}-1}2^{G_{\mathbf{M}}-1} + \ldots + \alpha_{g+1}2^{g+1} + 0\cdot 2^g + \alpha_{g-1}2^{g-1} + \cdots + \alpha_0 2^0, \tag{4.85}$$

$$i_{\mathbf{M}}^{(1)} = \alpha_{G_{\mathbf{M}}-1}2^{G_{\mathbf{M}}-1} + \ldots + \alpha_{g+1}2^{g+1} + 1\cdot 2^g + \alpha_{g-1}2^{g-1} + \cdots + \alpha_0 2^0. \tag{4.86}$$

The following Theorem is equivalent to say that GCK algorithm can be used KHT:

Theorem 4.5 If KHT projection values $y(N,i^{(n)},j)$ for $n = 0, 1$ as defined in (4.84)

are α-related at g with $i_{\mathbf{M}}^{(n)}$ represented in (4.85) and (4.86), then we have the followings:

$$\begin{aligned} y(N, i^{(0)}, j + R_2) &= y(N, i^{(0)}, j) - [y(N, i^{(1)}, j) + y(N, i^{(1)}, j + R_2)], \\ y(N, i^{(1)}, j + R_2) &= [y(N, i^{(0)}, j) - y(N, i^{(0)}, j + R_2)] - y(N, i^{(1)}, j), \end{aligned} \tag{4.87}$$

where R is the length of vector $\vec{\mathbf{s}}_r^{(i_r)}$ in (4.84), $R_2 = 2^{G_M - g - 1} R$.

Proof:

This proof is very similar to the proof for Theorem 4.2 in Appendix D. We first prove the computation in (4.87) when 2 α-related KHT basis vectors have the form of $\vec{\mathbf{s}}_l^{(i_l)^T} \otimes M^{(2)} \otimes \vec{\mathbf{s}}_r^{(i_r)^T}$, where $\vec{\mathbf{s}}_l^{(i_l)}$ has length L_2 and $\vec{\mathbf{s}}_r^{(i_r)}$ has length R_2. This special case has $N_{\mathbf{M}} = 2$ for WHT basis vector $\vec{\mathbf{m}}^{(N_{\mathbf{M}}, i_{\mathbf{M}}^{(n)})}$. In this form, we have:

$$\begin{bmatrix} \vec{\mathbf{v}}^{(N, i^{(0)})^T} \\ \vec{\mathbf{v}}^{(N, i^{(1)})^T} \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{s}}_l^{(i_l)^T} \otimes \vec{\mathbf{m}}^{(2,0)^T} \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \\ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes \vec{\mathbf{m}}^{(2,1)^T} \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \end{bmatrix} = \vec{\mathbf{s}}_l^{(i_l)^T} \otimes M^{(2)} \otimes \vec{\mathbf{s}}_r^{(i_r)^T}. \tag{4.88}$$

According to the definition of $\vec{\mathbf{v}}^{(N,i)}$ and $y(N, i, j)$ we have:

$$\begin{aligned} y(N, i^{(0)}, j) &= \left\{ \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(2,0)} \otimes \vec{\mathbf{s}}_r^{(i_r)} \right\}^T \vec{\mathbf{x}}_w^{(N,j)} \\ &= \left\{ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1] \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \right\} \vec{\mathbf{x}}_w^{(N,j)} \\ &= \left\{ \begin{bmatrix} \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1] & 0 \end{bmatrix} \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \right\} \vec{\mathbf{x}}_w^{(N+R_2,j)}, \\ y(N, i^{(0)}, j + R_2) &= \left\{ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1] \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \right\} \vec{\mathbf{x}}_w^{(N,j+R_2)} \\ &= \left\{ \begin{bmatrix} 0 & \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1] \end{bmatrix} \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \right\} \vec{\mathbf{x}}_w^{(N+R_2,j)}. \end{aligned} \tag{4.89}$$

Thus we have:

$$\begin{aligned} &\Delta_{N,j,R_2} \\ &= y(N, i^{(0)}, j) - y(N, i^{(0)}, j + R_2) \\ &= \left\{ \left(\begin{bmatrix} \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1] & 0 \end{bmatrix} - \begin{bmatrix} 0 & \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1] \end{bmatrix} \right) \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \right\} \vec{\mathbf{x}}_w^{(N+R_2,j)}, \\ &\quad y(N, i^{(1)}, j) + y(N, i^{(1)}, j + R_2) \\ &= \left\{ \left(\begin{bmatrix} \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ -1] & 0 \end{bmatrix} + \begin{bmatrix} 0 & \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ -1] \end{bmatrix} \right) \otimes \vec{\mathbf{s}}_r^{(i_r)^T} \right\} \vec{\mathbf{x}}_w^{(N+R_2,j)}. \end{aligned} \tag{4.90}$$

Let $\vec{\mathbf{s}}_l^{(i_l)}$ be represented as $[s_0\ s_1\ s_2\ \cdots, s_{L_2-1}]^T$. We have the followings for the

basis vectors in (4.90):

$$\begin{aligned}
&\left[\ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1] \quad 0\ \right] - \left[\ 0 \quad \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1]\ \right] \\
&= [s_0,\ s_0,\ s_1,\ s_1\ , s_2,\ s_2\ , \cdots,\ s_{L_2-1},\ s_{L_2-1},\ 0] \\
&\quad - [0,\ s_0,\ s_0,\ s_1,\ s_1,\ s_2,\ s_2\ \cdots,\ s_{L_2-1},\ s_{L_2-1}] \\
&= [s_0,\ 0,\ s_1 - s_0,\ 0\ , s_2 - s_1,\ 0, \cdots,\ 0,\ -s_{L_2-1}], \\
&\left[\ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ -1] \quad 0\ \right] + \left[\ 0 \quad \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ -1]\ \right] \\
&= [s_0,\ -s_0,\ s_1,\ -s_1,\ s_2,\ -s_2\ , \cdots,\ s_{L_2-1},\ -s_{L_2-1}\ 0] \\
&\quad + [0,\ s_0,\ -s_0,\ s_1,\ -s_1,\ s_2,\ -s_2\ , \cdots,\ s_{L_2-1},\ s_{L_2-1}] \\
&= [s_0,\ 0,\ s_1 - s_0,\ 0,\ s_2 - s_1,\ 0, \cdots,\ 0,\ -s_{L_2-1}] \\
&= \left[\ \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1] \quad 0\ \right] - \left[\ 0 \quad \vec{\mathbf{s}}_l^{(i_l)^T} \otimes [1\ 1]\ \right].
\end{aligned} \tag{4.91}$$

According to (4.91), basis vectors in (4.90) are equal and we have the followings when KHT has the form of $\vec{\mathbf{s}}_l^{(i_l)^T} \otimes M^{(2)} \otimes \vec{\mathbf{s}}_r^{(i_r)^T}$:

$$\Delta_{N,j,R_2} = y(N, i^{(0)}, j) - y(N, i^{(0)}, j + R_2) = y(N, i^{(1)}, j) + y(N, i^{(1)}, j + R_2). \tag{4.92}$$

Considering the general case, we have the followings:

$$\begin{aligned}
&\vec{\mathbf{v}}^{(N,i^{(0)})} = \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(N_{\mathrm{M}}, i_M^{(0)})} \otimes \vec{\mathbf{s}}_r^{(i_r)} \\
&= \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(2,\alpha_0)} \otimes \cdots \otimes \vec{\mathbf{m}}^{(2,\alpha_{g-1})} \otimes \vec{\mathbf{m}}^{(2,0)} \otimes \vec{\mathbf{m}}^{(2,\alpha_{g+1})} \otimes \cdots \otimes \vec{\mathbf{m}}^{(2,\alpha_{G_{\mathrm{M}}-1})} \otimes \vec{\mathbf{s}}_r^{(i_r)}, \\
&\vec{\mathbf{v}}^{(N,i^{(1)})} = \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(N_{\mathrm{M}}, i_M^{(1)})} \otimes \vec{\mathbf{s}}_r^{(i_r)} \\
&= \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(2,\alpha_0)} \otimes \cdots \otimes \vec{\mathbf{m}}^{(2,\alpha_{g-1})} \otimes \vec{\mathbf{m}}^{(2,1)} \otimes \vec{\mathbf{m}}^{(2,\alpha_{g+1})} \otimes \cdots \otimes \vec{\mathbf{m}}^{(2,\alpha_{G_{\mathrm{M}}-1})} \otimes \vec{\mathbf{s}}_r^{(i_r)}.
\end{aligned} \tag{4.93}$$

Denote $\vec{\mathbf{s}}_{l_2}$ and $\vec{\mathbf{s}}_{r_2}$ as follows:

$$\begin{aligned}
\vec{\mathbf{s}}_{l_2} &= \vec{\mathbf{s}}_l^{(i_l)} \otimes \vec{\mathbf{m}}^{(2,\alpha_0)} \otimes \cdots \otimes \vec{\mathbf{m}}^{(2,\alpha_{g-1})}, \\
\vec{\mathbf{s}}_{r_2} &= \vec{\mathbf{m}}^{(2,\alpha_{g+1})} \otimes \cdots \otimes \vec{\mathbf{m}}^{(2,\alpha_{G_{\mathrm{M}}-1})} \otimes \vec{\mathbf{s}}_r^{(i_r)}.
\end{aligned} \tag{4.94}$$

Then we have:

$$\begin{bmatrix} \vec{\mathbf{v}}^{(N,i^{(0)})^T} \\ \vec{\mathbf{v}}^{(N,i^{(1)})^T} \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{s}}_{l_2}^T \otimes \vec{\mathbf{m}}^{(2,0)^T} \otimes \vec{\mathbf{s}}_{r_2}^T \\ \vec{\mathbf{s}}_{l_2}^T \otimes \vec{\mathbf{m}}^{(2,1)^T} \otimes \vec{\mathbf{s}}_{r_2}^T \end{bmatrix} = \vec{\mathbf{s}}_{l_2}^T \otimes M^{(2)} \otimes \vec{\mathbf{s}}_{r_2}^T. \tag{4.95}$$

Since $\vec{\mathbf{v}}_N(i^{(n)})$ for $n = 0, 1$ in (4.95) are in the form shown in (4.88), we can use the relationship in (4.92) to derive the follows:

$$\Delta_{N,j,R_2} = y(N, i^{(0)}, j) - y(N, i^{(0)}, j + R_2) = y(N, i^{(1)}, j) + y(N, i^{(1)}, j + R_2). \quad (4.96)$$

Then we have this Theorem proved using (4.96) for deriving (4.87).

Chapter 5

The Orthogonal Haar transform and its Application in Full Search Equivalent Pattern Matching

5.1 Introduction

Many image processing applications require extracting information from images using filters, transforms or projection kernels. To extract information in a computationally efficient way, fast approaches such as integral image method [68], summed area table method [69], boxlet method [41], fast Walsh Hadamard transform (WHT) algorithms [1; 33], fast Gray-Code Kernels (GCK) algorithm [32], fast intersection kernel algorithm [70] and fast integral histogram algorithm [71] have facilitated research in tracking, pattern matching, texture mapping, face detection, object detection, etc. Motivated by these works, this chapter aims at improving the computational efficiency in extracting information from images.

5.1.1 Rectangle Sum and Integral Image

The sum of pixel values within a rectangle, which is named as rectangle sum in this chapter, is a commonly used feature for images. Rectangle sums and Haar-like features have been widely used for a lot of applications, e.g. object detection [68; 72–74], object classification [75], pattern matching [26; 31; 76], feature point based image matching [20] and texture mapping [69]. Porikli utilizes the integral image method to design a fast algorithm for computing the histogram in [71]. Since the work in [69] from 1984, it has been considered that at least 3 additions are required to obtain the rectangle sum using the summed area table in [69; 77] or the integral image method in [68]. Note that one subtraction is considered to be one addition regarding the computational complexity in this chapter.

This section illustrates the integral image following [68; 76; 77]. The summed area table in [69] is very similar to the integral image. Denote the pixel value at position (j_1, j_2) of a $J_1 \times J_2$ image as $x(j_1, j_2)$, where $0 \leq j_1 \leq J_1 - 1$ and $0 \leq j_2 \leq J_2 - 1$. The integral image $ii(j_1, j_2)$ is defined as:

$$ii(j_1, j_2) = \sum_{u=0}^{j_1-1} \sum_{v=0}^{j_2-1} x(u, v). \tag{5.1}$$

It is reported by Viola and Jones that the integral image in (5.1) can be constructed using two additions per pixel [68]. As shown in Fig. 5.1, a rectangle in an image is specified by:

$$rect = (j_1, j_2, N_1, N_2), \tag{5.2}$$

where j_1 and j_2 denote the upper left position of the rectangle, N_1 and N_2 denote the size of the rectangle, $0 \leq j_1, j_1 + N_1 \leq J_1 - 1; 0 \leq j_2, j_2 + N_2 \leq J_2 - 1;\ N_1, N_2 > 0$.

Denote $rs(rect)$ as the sum of pixels within the rectangle specified by the $rect$ in (5.2). The integral image method in [68] computes $rs(rect)$ from integral image using 3 additions as follows:

$$\begin{aligned} rs(rect) &= \sum_{u=j_1}^{j_1+N_1-1} \sum_{v=j_2}^{j_2+N_2-1} x(u, v) \\ &= ii(j_1 + N_1, j_2 + N_2) + ii(j_1, j_2) \\ &\quad - ii(j_1, j_2 + N_2) - ii(j_1 + N_1, j_2). \end{aligned} \tag{5.3}$$

Lienhart and Maydt propose an algorithm that computes the sum of pixels within the 45° rotated rectangle by 3 additions in [77] using similar method as the integral image method.

5.1.2 Overview

This chapter proposes a fast algorithm that computes the sum of pixels in a rectangle by one addition using the proposed strip sum. Then we propose the Orthogonal Haar

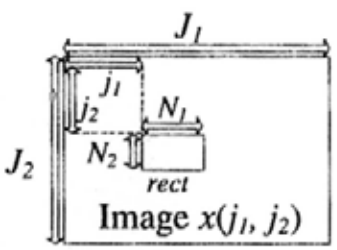


Figure 5.1: *Examples of rectangle rect, where* $rect = (j_1, j_2, N_1, N_2)$, j_1 *is the horizontal position and* j_2 *is the vertical position of the upper left corner,* N_1 *is the width and* N_2 *is the height of the rectangle.* J_1 *is the width and* J_2 *is the height of the image.*

transform (OHT) which can be computed using strip sum. When computed on sliding windows, the proposed algorithm requires $O(\log u)$ additions per pixel to compute u OHT basis vectors. OHT is then used for FS equivalent pattern matching. Experimental results show that pattern matching using OHT is faster than existing FS equivalent algorithms. The method was initially introduced in [78]. We expand this idea here, as well as introduce more experimental results and theoretical analysis on the relationship between OHT and other transforms.

The chapter is organized as follows. Section 5.2 presents the strip sum and then analyzes the computation and memory required by strip sum for computing rectangle sum. Section 5.3 proposes the OHT and analyzes its computational complexity. Section 5.4 gives experimental results. Finally, Section 5.5 presents conclusions.

5.2 The Fast Algorithm for Computing Rectangle Sum

In this section, the fast algorithm for computing rectangle sum is introduced and analyzed regarding computational complexity and memory requirement.

5.2.1 Computation of Rectangle Sum by Strip Sum

Define horizontal strip sum $hss(j_1, j_2, N_2)$ as:

$$hss(j_1, j_2, N_2) = \sum_{u=0}^{j_1-1} \sum_{v=j_2}^{j_2+N_2-1} x(u, v). \tag{5.4}$$

Hence, $hss(j_1, j_2, N_2)$ has the upper right corner $(j_1 - 1, j_2)$ and height N_2. Figure 5.2 shows the $hss(j_1 + N_1, j_2, N_2)$ and $hss(j_1, j_2, N_2)$ defined in (5.4). $hss(j_1, j_2, N_2)$ can be obtained by one addition per pixel as follows using the integral image $ii(j_1, j_2)$ defined in (5.1):

$$\begin{aligned} hss(j_1, j_2, N_2) &= \sum_{u=0}^{j_1-1} \sum_{v=j_2}^{j_2+N_2-1} x(u, v) \\ &= \sum_{u=0}^{j_1-1} \sum_{v=0}^{j_2+N_2-1} x(u, v) - \sum_{u=0}^{j_1-1} \sum_{v=0}^{j_2-1} x(u, v) \\ &= ii(j_1, j_2 + N_2) - ii(j_1, j_2). \end{aligned} \tag{5.5}$$

We can compute rectangle sums $rs(j_1, j_2, N_1, N_2)$ and $rs(j_1, j_2, N_1', N_2)$ $(N_1 \neq N_1')$

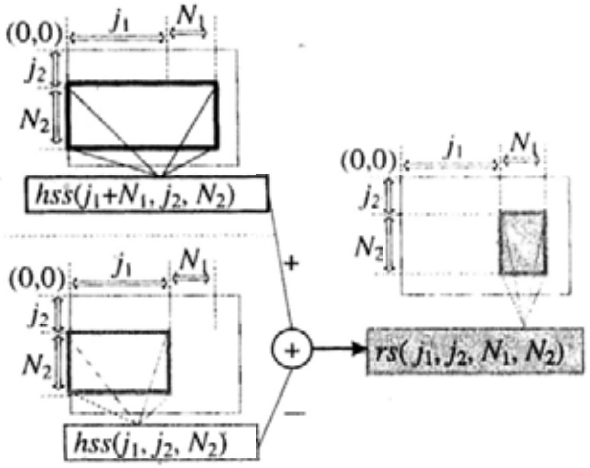


Figure 5.2: *Strip sum and rectangle sum on the image. Only one addition is required to compute $rs(j_1, j_2, N_1, N_2)$ from the data structure hss using (5.6).*

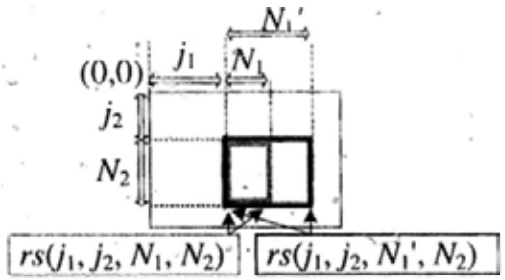


Figure 5.3: *Rectangle sums sharing the same height N_2. The two rectangle sums can use the same strip sum for computation.*

from strip sums as follows using (5.3) and (5.5):

$$\begin{aligned} & rs(j_1, j_2, N_1, N_2) \\ & = [ii(j_1 + N_1, j_2 + N_2) - ii(j_1 + N_1, j_2)] \\ & \quad - [ii(j_1, j_2 + N_2) - ii(j_1, j_2)] \\ & = hss(j_1 + N_1, j_2, N_2) - hss(j_1, j_2, N_2), \\ & rs(j_1, j_2, N_1', N_2) \\ & = hss(j_1 + N_1', j_2, N_2) - hss(j_1, j_2, N_2). \end{aligned} \tag{5.6}$$

As shown in (5.6), only one addition is required to compute $rs(j_1, j_2, N_1, N_2)$ from the strip sums $hss(j_1 + N_1, j_2, N_2)$ and $hss(j_1, j_2, N_2)$. Fig. 5.2 shows this computation.

The $rs(j_1, j_2, N_1, N_2)$ and $rs(j_1, j_2, N_1', N_2)$ in (5.6) have the same height N_2 but have different width ($N_1 \neq N_1'$). They both use the $hss(j_1, j_2, N_2)$ for computation. Fig. 5.3 shows the relationship between $rs(j_1, j_2, N_1, N_2)$ and $rs(j_1, j_2, N_1', N_2)$. Thus one strip sum $hss(j_1, j_2, N_2)$ is utilized for computation of two rectangle sums having sizes $N_1 \times N_2$ and $N_1' \times N_2$ in (5.6). In general, if the data structure $hss(j_1, j_2, N_2)$

has been provided on all pixel locations (j_1, j_2), we can use $hss(j_1, j_2, N_2)$ to obtain rectangle sums having any width N_1 and the fixed height N_2 at any pixel position by one addition. This procedure of using strip sum for computing rectangle sum is shown in Table 5.1. Strip sum is also applicable for 45° rotated rectangles introduced in [77].

1. $Temp_{hss}$ is a 1-D array buffer of size J_1. N_2 is a fixed value. j_2 is the row index and j_1 is the column index
2. for j_2= 0 to J_2 − 1
3. Compute the $hss(j_1, j_2, N_2)$ for $0 \le j_1 \le J_1$-1 at the j_2th row from integral image using (7) and store them into $Temp_{hss}$.
4. for j_1=0 to J_1-1
5. Compute $rs(j_1, j_2, N_1, N_2)$ using (8) for rectangles having any N_1. The $hss(j_1, j_2, N_2)$ and $hss(j_1+N_1, j_2, N_2)$ in (8) are stored in $Temp_{hss}$.
6. end of for j_1=0 to J_1-1

Table 5.1: *Pseudo-code showing the utilization of strip sum for computing rectangle sums sharing the same height N_2.*

hss is used for rectangle sums having the same height N_2 in (5.6). 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:

$$vss(j_1, j_2, N_1) = \sum_{u=j_1}^{j_1+N_1-1} \sum_{v=0}^{j_2-1} x(u, v). \tag{5.7}$$

And we can use the same vss for computing rectangle sums having any height N_2 and fixed width N_1 as follows:

$$\begin{aligned} &rs(j_1, j_2, N_1, N_2) \\ &= vss(j_1, j_2 + N_2, N_1) - vss(j_1, j_2, N_1), \\ &\text{where } vss(j_1, j_2, N_1) = ii(j_1 + N_1, j_2) - ii(j_1, j_2). \end{aligned} \tag{5.8}$$

5.2.2 Computational Complexity Analysis

The overall algorithm and the computation required by strip sum for computing one rectangle sum is as follows:

1. prepare the integral image by 2 additions per pixel;
2. prepare the strip sum using (5.5) by 1 addition per pixel;
3. obtain the rectangle sum from strip sum using (5.6) by 1 addition per pixel.

1. prepare the integral image by 2 additions and 4 memory fetch operations (M-ops) per pixel;
2. prepare the $\min\{Num_W, Num_H\}$ strip sums using (5.5) or (5.8) by $\min\{Num_W, Num_H\}$ additions and $2\min\{Num_W, Num_H\}$ M-ops per pixel;
3. obtain the r rectangle sum from strip sum using (5.6) or (5.8) by r additions and $2r$ M-ops per pixel.

Table 5.2: *The steps and number of operations required by strip sum method to obtain r rectangle sums.*

	Box	Integral image	Strip sum
Adds	$4r$	$2+3r$	$2+\min\{Num_W, Num_H\}+r$
M-ops	$6r$	$4+4r$	$4+2\min\{Num_W, Num_H\}+2r$

Table 5.3: *The additions (*adds*) and memory fetch operations (*M-op*) per pixel required for computing r rectangle sums having Num_H different heights and Num_W different widths.*

For general case, we suppose there are r different sizes of rectangle sums computed on each pixel position. Suppose these r rectangle sums have Num_W different widths and Num_H different heights. As illustrated in Table 5.1, one horizontal strip sum is required for rectangle sums sharing the same height. Hence, Num_H horizontal strip sums are required for computing r rectangle sums having Num_H different heights. Alternatively, we can compute these r rectangle sums using Num_W vertical strip sums. Therefore, $\min\{Num_W, Num_H\}$ strip sums are required for computing these r rectangle sums. The overall algorithm and the computation required for computing r rectangle sums is shown in Table 5.2. In summary, the strip sum method requires $2 + \min\{Num_W, Num_H\} + r$ additions and $4 + 2\min\{Num_W, Num_H\} + 2r$ memory fetch operations (M-ops) per pixel for computing r rectangle sums.

The box technique in [47] requires $4r$ additions and $6r$ M-ops per pixel for computing r rectangle sums. The integral image method in [68] requires $2 + 3r$ additions and $4 + 4r$ M-ops per pixel, where 2 additions and 4 M-ops are used for preparing the integral image, $3r$ additions and $4r$ M-ops are used for computing r rectangle sums from integral image using (5.3). The computational complexity for different methods is compared in Table 5.3.

5.2.3 Buffering Strip Sum

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

We use the computation of rectangle sums having the same height N_2 but different widths as the example for illustrating the buffering strategy. The procedure of memory usage is described in Table 5.1. Suppose the rectangle sums at the 0th row are obtained. When we use (5.6) for computing rectangle sums $rs(j_1, 0, N_1, N_2)$ having any width N_1 at row 0, we need only the strip sums $hss(j_1, 0, N_2)$ and $hss(j_1 + N_1, 0, N_2)$ at row 0, but need not the strip sum at other rows. Thus we can compute the strip sums at row 0 and store them using a buffer $Temp_{hss}$ having size J_1 at Step 3 in Table 5.1, but need not store the strip sums at other rows. As the row index j_2 increases from 0 to 1, the strip sums in row 0 is not required any more. And we can reuse the buffer $Temp_{hss}$ for storing trip sums in row 1, which is then used for computing rectangle sums in row 1. Therefore, size J_1 memory for $Temp_{hss}$ is required to store strip sums in this example as j_2 increases from 0 to $J_2 - 1$. In general, if r rectangle sums having Num_H different heights are computed, size $J_1 Num_H$ memory is required to store the Num_H strip sum buffers, where $Num_H \leq J_2$. Thus the memory required by the strip sum using buffering strategy is smaller than or equal to image size $J_1 J_2$.

5.3 The Orthogonal Haar Transform

5.3.1 The Proposed Orthogonal Haar Transform

Fig. 5.4 shows the proposed 2D 4×4 and 8×8 OHT. The OHT for other sizes can be similarly derived. Normally, the basis for 2D image is called basis image. Since 2D image can be represented by 1D vector, e.g. 2D candidate window and pattern are represented as 1D vectors $\vec{\mathbf{x}}_w^{(N,j)}$ and $\vec{\mathbf{x}}_t^{(N)}$ respectively, basis image is called basis vector in this chapter so that the 2D window represented by 1D vector is projected onto basis vector instead of basis image. It is easy to see that the OHT basis vectors in Fig. 5.4 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

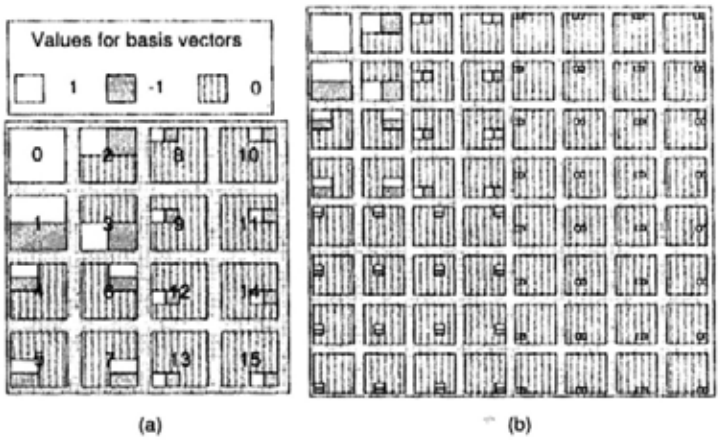


Figure 5.4: *(a): The 2D 4 × 4 OHT basis; (b): the 2D 8 × 8 OHT basis. White represents value +1, grey represents value −1 and vertical strips represent value 0. The numbers for 4 × 4 OHT basis denote the order when they are computed in pattern matching.*

shown in Table 1.1.

The GCK algorithm requires $2u$ additions for obtaining u GCK projection values. If we directly use the integral image for computing Haar-like features as the implementation in OpenCV [53], 7 additions are required to obtain a Haar-like feature and about $7u$ additions are required for obtaining u OHT projection values. Hence, direct use of integral image method makes OHT less efficient than WHT and GCK in computing transformation. In the followings, we propose a fast OHT algorithm that requires $O(\log u)$ additions for obtaining u OHT projection values.

5.3.2 The Fast OHT Algorithm

The 2D 4 × 4 OHT in Fig. 5.4 is used as an example to illustrate the fast OHT algorithm. In the proposed algorithm, the 16 4 × 4 OHT basis vectors are considered to be 1 rectangle sum, i.e. basis vector 0, and 4 Haar-like features having the following different sizes:

1. basis vector 1 has size 4 × 4;
2. basis vectors 2 and 3 have the same size 4 × 2, i.e. width 4 and height 2, but different positions;
3. basis vectors 4 to 7 have the same size 2 × 2 but different positions;
4. basis vectors 8 to 15 have the same size 2 × 1 but different positions.

Since the basis vectors having the same size but different positions are the same Haar-like feature having different positions on the image, they can be simultaneously obtained by computing one Haar-like feature in a sliding window manner on the image. Haar-like features can be obtained from rectangle sums by 1 addition. Thus the 4 Haar-like features having different sizes can be obtained from the rectangle sums by 4 additions. The 16 basis vectors are computed from $r = 5$ rectangle sums having different sizes: 4×4, 4×2, 2×2, 2×1 and 1×1, where we have $Num_H = 3$ different heights: 4, 2 and 1. According to the analysis in Table 5.3, we need $2 + 3 + 5 = 10$ additions for obtaining the $\min\{Num_W, Num_H\} = 3$ strip sums and $r = 5$ 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 preparing strip sums and rectangle sums.

As another example, the 64 8×8 OHT basis vectors in Fig. 5.4 are considered to be 1 rectangle sum and 6 Haar-like features having different sizes. The 64 basis vectors are computed from $r = 7$ rectangle sums having different sizes: 8×8, 8×4, 4×4, 4×2, 2×2, 2×1 and 1×1, where we have $Num_H = 4$ different heights: 8, 4, 2 and 1. From the 16 4×4 OHT basis vectors to the 64 8×8 OHT basis vectors, Num_H increases by 1, the number of Haar-like features and the number of rectangle sums having different sizes increase by 2.

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 $\log_2 u$ Haar-like features having different sizes. There are $r = \log_2 u + 1$ 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 $Num_H = 0.5log_2u + 1$. The steps for computing OHT and the corresponding number of operations required are shown in Table 5.4. In summary, the proposed method requires $4 + 2.5 \log_2 u$ additions for obtaining u OHT projection values. When $u = 16$, we have the example for the 4×4 OHT.

As for memory required in computing OHT, only the strip sum having one height need be stored. For example, when we compute the $0th$ 4×4 OHT basis vector, the horizontal strip sum having height 4 is stored. Then we compute the $1st$ basis vector using the horizontal strip sum having height 2, while the horizontal strip sums having height 4 for the $0th$ basis vector are not required and are not stored any more. To

1. prepare the integral image by 2 additions per pixel;
2. obtain the $Num_H = 0.5log_2u + 1$ strip sums from integral image using (5.5) by $0.5log_2u + 1$ additions per pixel;
3. obtain the $r = \log_2 u + 1$ rectangle sums from strip sum using (5.6) by $\log_2 u + 1$ additions per pixel;
4. obtain the $\log_2 u$ Haar-like features from rectangle sums by $\log_2 u$ additions per pixel.

Table 5.4: *The steps and number of operations required to obtain u OHT projection values.*

further save memory, we can use the buffering strategy in Section 5.2.3 and require J_1 memory for storing strip sum.

5.3.3 OHT for Pattern Matching

Since OHT bases have varying norms, normalization is required for obtaining SSD. 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 the remaining candidates, i.e. $||\mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_l^{(N)} - \mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_w^{(N,j)}||^2$ for $\vec{\mathbf{x}}_w^{(N,j)} \in set_{can}$, but not for OHT on the entire image, i.e. $\mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_w^{(N,j)}$ for $j = 0, 1, 2, \ldots, W-1$. For $u = 4^n$ OHT bases, there are $\log_2 u$ different normalization factors. In worst case, the normalization requires $\log_2 u$ shifts per pixel when no candidate is rejected at all. This normalization procedure requires few computation in practical cases where many candidates are rejected in early stages.

OHT can be computed in two ways: sliding window or random access. We choose one of the two ways that require less computation for pattern matching. For example, if we want to project 500 candidate windows onto the first basis in a 256×256 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 256^2 windows instead of for the remaining 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. The corners method introduce in [59] for WHT can compute the WHT for the remaining 500 candidate windows. The corners method requires $O(N)$ additions for obtaining one WHT basis while the OHT algorithm require $O(1)$ additions for one

OHT basis.

5.3.4 Comparison of OHT with Other Transforms

The following theorem describes the links between WHT and OHT:

Theorem 5.1 If the 2D $N_1 \times N_2$ WHT bases are in the same order as OHT bases, then: 1) the subspace spanned by the first $u = 4^n, n = 0, 1, ...$ WHT bases is equal to the subspace spanned by the first u OHT bases; 2) the first u orthonormal WHT bases and the first u orthonormal OHT bases extract the same energy from any input data; 3) the computation for the u WHT bases requires $3u/2 + 1$ additions per window while the computation for the u OHT bases requires $4 + 2.5 \log_2 u$ additions.

The proof for the theorem above is provided in the appendix. We use the 4×4 OHT and 4×4 WHT example in Fig. 5.5 for illustration. Let the ith 2D WHT basis vector be represented as $\vec{\mathbf{v}}_{WHT}^{(i)}$. Let the ith 2D OHT basis vector be represented as $\vec{\mathbf{v}}_{OHT}^{(i)}$. We have the follows for the first $u = 4$ WHT and OHT basis vectors in Fig. 5.5:

$$\begin{bmatrix} \vec{\mathbf{v}}_{WHT}^{(0)^T} \\ \vec{\mathbf{v}}_{WHT}^{(1)^T} \\ \vec{\mathbf{v}}_{WHT}^{(2)^T} \\ \vec{\mathbf{v}}_{WHT}^{(3)^T} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} \vec{\mathbf{v}}_{OHT}^{(0)^T} \\ \vec{\mathbf{v}}_{OHT}^{(1)^T} \\ \vec{\mathbf{v}}_{OHT}^{(2)^T} \\ \vec{\mathbf{v}}_{OHT}^{(3)^T} \end{bmatrix}, \tag{5.9}$$

where the first 4 WHT bases can be linearly represented by the first 4 OHT bases and vice versa. Thus the subspace spanned by the first 4 orthogonal WHT bases is equal to the subspace spanned by 4 OHT bases. The WHT bases are orthogonal to each other and can be normalized to be orthonormal bases, so are the OHT bases. Thus the energy extracted from input data by the first 4 orthonormal WHT bases is equal to that extracted by the first 4 orthonormal OHT bases. As for computational complexity, the fastest WHT algorithm in [33] requires $3u/2 + 1 = 25$ additions to compute the $u = 16$ WHT bases, the proposed fast OHT algorithm requires 14 additions to compute the 16 OHT bases.

The conventional Haar transform (HT) is different from the OHT proposed in this chapter. Fig. 5.5 shows 2D 4×4 OHT, HT and WHT. In appendix A, we prove that the first $u = 4^n$ conventional HT bases and the first u proposed OHT bases span the same subspace. However, it is more efficient computing the OHT than computing the conventional HT. For example, OHT bases 2 and 3 can be obtained at the same time when computed on sliding windows. However, the conventional HT bases 2 and 3 are

required to be computed independently and the computation of HT basis 3 is more complex than OHT basis 3.

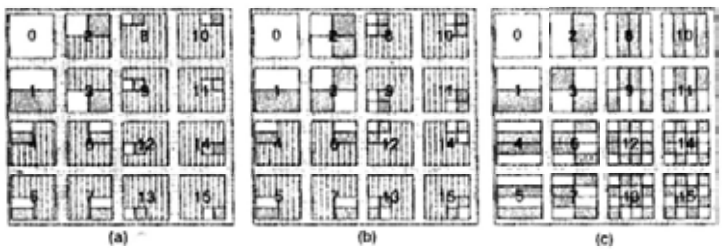


Figure 5.5: *2D 4 × 4 transforms: (a) the proposed OHT, (b) conventional Haar transform, (c) WHT. White represents +1, grey represents −1 and green vertical strips represent 0. The numbers for 4 × 4 OHT basis denote the order when they are computed in pattern matching.*

The elements in OHT only contain 1, -1 and 0. The elements in GCK [32] can be real numbers. GCK [32] and generalized GCK [51] are better than OHT in representing a larger set of basis vectors that can be computed efficiently. Efficient computation of GCK and generalized GCK requires that they are computed in sliding window manner. OHT is more efficient than GCK when computed on sliding windows. Another advantage of OHT over GCK and generalized GCK is in random access, i.e. when computing transformation for isolated windows as analyzed in Section 5.3.3.

5.4 Experimental Results

This section evaluates the performance of pattern matching using OHT by comparing it with FS and the other FS equivalent algorithms in pattern matching. All of the experiments are implemented on a 2.13GHz PC using C on windows XP system with compiling environment VC 6.0. Sum of squared difference (SSD) is used for measuring the dissimilarity between pattern and candidate windows.

5.4.1 Dataset and Algorithms Used for Pattern Matching Experiments

The fast algorithms compared with FS are as follows:

1. WHT: the WHT algorithm in [1];
2. GCK: the GCK algorithm for WHT in [32];
3. IDA: the recently proposed IDA algorithm in [31];

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

Table 5.5: *Datasets and corresponding sizes of images and patterns used in the experiments.*

4. OHT_I: the integral image for OHT;

5. OHT_S: the strip sum for OHT.

The code for WHT is available online [57] and the code for IDA is provided by the authors of [31]. The parameters for WHT use the default values in [57] and those for IDA are chosen according to [31]. For GCK, we choose the sequency ordered WHT and the code is based on the code used for motion estimation in [2]. In the experiments, we examine the overall computational efficiency of different algorithms instead of any single step if not specified. Thus all preprocessing, e.g. calculating integral image and strip sum, are included for evaluation.

As explained in [1], when the percentage of remaining candidate windows is smaller than certain number, denoted as ϵ, it is more efficient to use FS for finding the matched windows instead of using transformation. In the experiments, the default setting is $\epsilon = 0.02\%$ for OHT. According to the authors' source code for WHT in [57], we set $\epsilon = 2\%$ as default value for WHT and GCK. We will give variations of ϵ in Section 5.4.4.

Table 5.5 shows the 6 datasets used for evaluating the performance of the compared algorithms. The dataset includes different sizes of patterns and images. Our experiments include a total of 120 images chosen among three databases: MIT [54], medical [55], and remote sensing [56]. 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 contain radiographs and Landsat satellite images. The 120 images have 4 resolutions which are 160×120, 320×240, 640×480 and 1280×960. Each resolution has 30 images. The OpenCV function 'cvResize' with linear interpolation is used for producing the desired resolution of 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 5.5 were formed. Each dataset has 300 image-pattern pairs. Datasets $S1$ to $S4$ are the same as those in [31]. 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:

$$T = 1.1 \cdot SSD_{min} + N_1 N_2, \tag{5.10}$$

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

5.4.2 Experiment 1 – Pattern Matching Algorithms on Different Sizes

In this experiment, we compare the speed-ups yielded by the considered algorithms in pattern matching on datasets $S1 - S4$ which have different sizes of image-pattern pairs. The speed-up in execution time or speed-up in operation for 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 speed-up in execution time 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 larger is the speed-up, the faster is the algorithm. There are 300 image-pattern pairs tested for each dataset. The speed-ups in execution time yielded by GCK, IDA and OHT over the FS in pattern matching for each dataset are shown in Fig. 5.6. It can be seen that OHT outperforms the other compared algorithms for the 4 datasets. To compute OHT, strip sum requires about 50% the execution time of integral image. Pattern matching using strip sum for OHT requires about 50%-60% the time of that using integral image for OHT.

5.4.3 Experiment 2 – 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

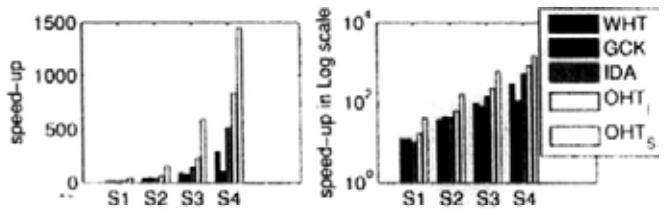


Figure 5.6: *Speed-up in execution time over FS on datasets S1 – S4 for different algorithms measured by normal scale (Left) and log scale (right). The bars for each dataset from left to right correspond to algorithms WHT, GCK, IDA, OHT_I and OHT_S.*

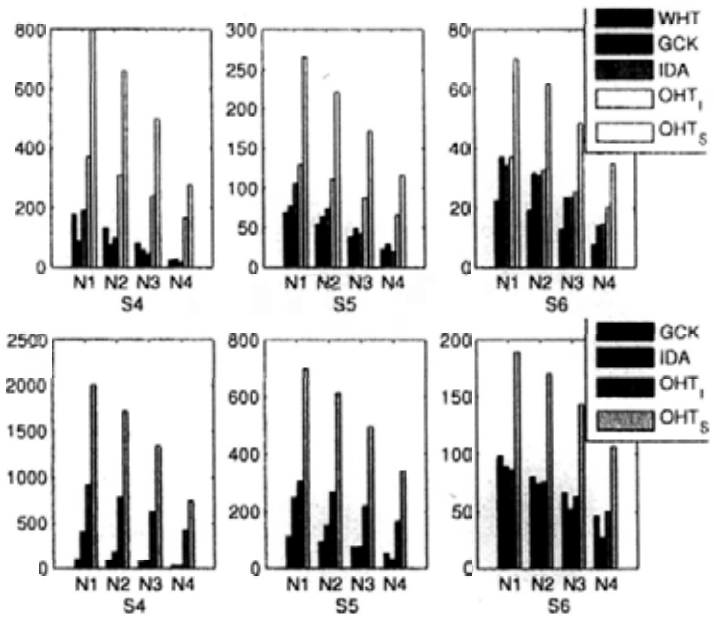


Figure 5.7: *Speed-ups in execution time (upper row) and speed-ups in number of operations (bottom row) yielded by different algorithms over FS for different noise levels and sizes of image-pattern pairs in pattern matching.*

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. 5.7 shows the speed-ups in execution time and speed-ups in number of operations yielded by examined fast algorithms over FS in different pattern sizes and different noise levels. It can be seen that OHT is the fastest. Fig. 5.8 shows the speed-ups in execution time and speed-ups in number of operations yielded by OHT over IDA and GCK on dataset $S4$. As shown in Fig. 5.8 on dataset $S4$, the speed-ups in execution time yielded by OHT over IDA and GCK are about 4 to 15 and 8 to 10 respectively, the speed-ups in number of operations yielded by OHT over IDA and GCK are about 5 to 24 and 19 to 24 respectively. The speed-up in number of operations for WHT is

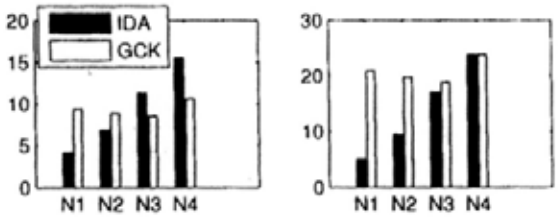


Figure 5.8: *Speed-up of OHT over IDA and GCK at 4 noise levels for dataset S4 in pattern matching.* Left: *speed-up in execution time;* right: *speed-up in number of operations.*

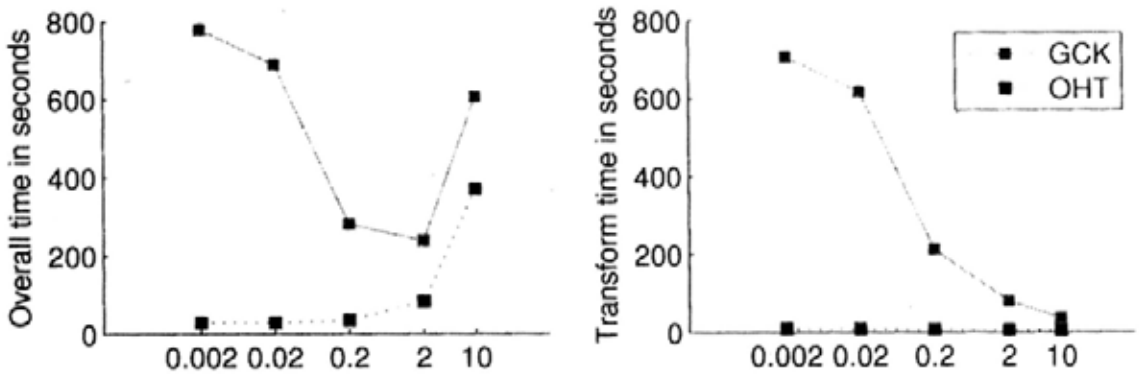


Figure 5.9: *(a) The overall execution time in seconds as a function of ϵ on the left figure and (b) the transformation time in seconds as a function of ϵ on the right figure. Experiments are done on dataset S4 with Noise G(4). ϵ denotes the percentage of remaining window below which FS is used for pattern matching, e.g. 2 and 10 in the X axis correspond to 2% and 10% respectively.*

not provided because the bottom up approach of WHT method in [1] is too complex for analysis.

5.4.4 Experiment 3 – Influence of Parameter ϵ

As explained in [1], when the percentage of remaining candidate windows is smaller than certain number ϵ, it is more efficient to directly use SSD for finding the matched windows instead of using transformation. Fig. 5.9(a) shows the influence of ϵ for both GCK and OHT on dataset $S4$ with Gaussian noise $G(4)$. It can be seen that 2% is better for GCK while 0.02% is better for OHT. Fig. 5.9(b) shows that the computation of OHT is obviously faster than the computation of GCK for different situations of ϵ.

5.4.5 Experiment 4 – Energy Packing Ability of OHT and WHT

Experiments 1, 2 and 3 evaluate the execution time required for the whole pattern matching procedure. Here, we evaluate the energy packing ability of OHT and WHT. As analyzed in [1], the computational efficiency of transform domain pattern matching

is dependent on two factors: 1) the rejection power of projection values, which is dependent on the energy packing ability of transformation; 2) the cost of computing transformation. In summary, transform domain pattern matching requires that the transformation should be computationally efficient in packing energy.

We examine the energy packing ability of OHT and WHT on dataset $S1$ in Table 5.5. This dataset contains 300 image-pattern pairs, i.e. 4,567,500 window-pattern pairs of size 16×16. The SSD between the window and pattern were computed. The percentage of energy packed by the first u basis vectors is measured by:

$$PER^{(u)} = E\left[\frac{||\mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_t^{(N)} - \mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_w^{(N,j)}||^2}{||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||^2}\right], \tag{5.11}$$

where $||\mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_t^{(N)} - \mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_w^{(N,j)}||^2$ is the partial energy packed by transformation $\mathbf{V}^{(u\times N)}$ and $||\vec{\mathbf{x}}_t^{(N)} - \vec{\mathbf{x}}_w^{(N,j)}||^2$ is the actual energy.

Fig. 5.10(a) shows the $PER^{(u)}$ in (5.11) as a function of the number of bases u. The results are the average over the 4,567,500 pattern-window pairs. It can be seen that OHT is close to WHT in packing energy. These results, however, do not exhibit the runtime required in extracting energy. Fig. 5.10(b) shows the $PER^{(u)}$ in (5.11) as a function of the number of operations per pixel required to obtain the partial energy $||\mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_t^{(N)} - \mathbf{V}^{(u\times N)}\vec{\mathbf{x}}_w^{(N,j)}||^2$. It can be seen that OHT can extract energy from input data using much fewer number of additions compared with WHT.

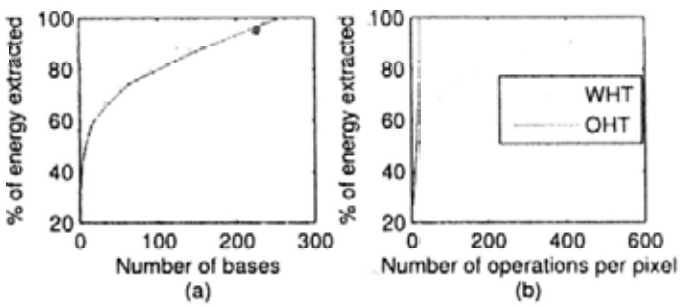


Figure 5.10: *Transformation using WHT and OHT. WHT is in sequency order and OHT is in the order illustrated in Fig. 5.4. The energy is given as the average percentage of the actual energy between pattern and window. All values are the average over 4,567,500 window-pattern pairs.*

5.5 Summary

This chapter 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 [42; 48; 77]. Then we propose to use the OHT for sliding transform. Existing fast algorithms for computing transformation on sliding windows [1; 32; 33] require $O(u)$ additions for projecting input data having size N onto u 2D basis. The OHT can project input data onto u 2D basis by $O(\log u)$ additions using the strip sum. For example, when $u = N$, OHT requires $O(\log N)$ additions for projecting input data having size N onto N basis vectors. 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.

5.6 Appendix A: Proof of Theorem 5.1

Theorem 5.1: If the 2D $N_1 \times N_2$ WHT bases are in the same order as OHT bases, then: 1) the subspace spanned by the first $u = 4^n, n = 0, 1, ...$ WHT bases is equal to the subspace spanned by the first u OHT bases; 2) the first u orthonormal WHT bases and the first u orthonormal OHT bases extract the same energy from any input data; 3) the computation for the u WHT bases requires $3u/2 + 1$ additions per window while the computation for the u OHT bases requires $4 + 2.5 \log_2 u$ additions.

Proof: For 1D HT and WHT, it is proved in [79] that the first $u = 2^n, n = 0, 1, 2, \ldots$ 1D WHT basis vectors can be linearly represented by the first u 1D HT basis vectors when these basis vectors are ordered by their frequencies (number of sign changes). Since both HT and WHT are separable transformations, the first $u = 4^n, n = 0, 1, 2, \ldots$ 2D WHT bases can be linearly represented by the first u 2D HT bases when ordered

as shown in Fig. 5.5. Thus we have:

$$\mathbf{V}_{WHT}^{(u\times N)} = \mathbf{D}_{WHT,HT}^{(u\times u)}\mathbf{V}_{HT}^{(u\times N)}, \tag{5.12}$$

where $\mathbf{V}_{HT}^{(u\times N)} = [\vec{\mathbf{v}}_{HT}^{(0,N)}, \vec{\mathbf{v}}_{HT}^{(1,N)}, \ldots, \vec{\mathbf{v}}_{HT}^{(u-1,N)}]^T$ contains the first u 2D HT bases, $\mathbf{V}_{WHT}^{(u\times N)}$ contains the first u 2D WHT bases, rank u matrix $\mathbf{D}_{WHT,HT}^{(u\times u)}$ represents the linear relationship between HT and WHT. It is easy to find that the first $u = 4^n$ conventional HT bases can be represented by the proposed OHT bases as follows:

$$\begin{aligned}\mathbf{V}_{HT}^{(u\times N)} &= \mathbf{D}_{HT,OHT}^{(u\times u)}\mathbf{V}_{OHT}^{(u\times N)} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 \\ 0 & \mathbf{I}_1 & 0 & 0 & 0 & \ldots & 0 & 0 \\ 0 & 0 & \mathbf{M}_{OHT}^{(1)} & 0 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 0 & \mathbf{I}_4 & 0 & \ldots & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{M}_{OHT}^{(4)} & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \ldots & \mathbf{I}_{\frac{N}{4}} & 0 \\ 0 & 0 & 0 & 0 & 0 & \ldots & 0 & \mathbf{M}_{OHT}^{(\frac{N}{4})} \end{bmatrix} \mathbf{V}_{OHT}^{(u\times N)}, \\ \text{where } \mathbf{M}_{OHT}^{(n)} &= \mathbf{I}_n \otimes \begin{bmatrix} \mathbf{I}_n \otimes [1 \quad 1] \\ \mathbf{I}_n \otimes [1 \quad -1] \end{bmatrix},\end{aligned} \tag{5.13}$$

$\otimes$ is the Kronecker product, $\mathbf{I}_n$ is the $n \times n$ identity matrix. Considering (5.12) and (5.13), we have:

$$\mathbf{V}_{WHT}^{(u\times N)} = \mathbf{D}_{WHT,HT}^{(u\times u)}\mathbf{D}_{HT,OHT}^{(u\times u)}\mathbf{V}_{OHT}^{(u\times N)}, \tag{5.14}$$

where both $\mathbf{D}_{WHT,HT}^{(u\times u)}$ and $\mathbf{D}_{HT,OHT}^{(u\times u)}$ has full rank u, $\mathbf{V}_{OHT}^{(u\times N)}$ contains the first u 2D OHT bases. Thus the subspace spanned by the u WHT bases is equal to the subspace spanned by the u OHT bases. The WHT bases are orthogonal to each other and can be normalized to be orthonormal, so are OHT bases. Therefore, we can conclude that: 1) the subspace spanned by the first u WHT basis vectors can be spanned by the first u OHT basis vectors; 2) the energy extracted by the first u orthonormal WHT bases can also be extracted by the first u OHT orthonormal bases; 3) the algorithm in [33] requires $3u/2 + 1$ additions per window for obtaining u 2D WHT bases, the approach introduced in Section 5.3.1 requires $4 + 2.5\log_2 u$ additions per window for obtaining u 2D OHT bases.

Chapter 6

Fast Transform Domain Linear Support Vector Machine using Pruning

6.1 Introduction

Support Vector Machines (SVMs) are widely used learning systems in real world pattern recognition and data mining applications such as object recognition, human detection, text categorization, hand-written character recognition, image classification and bioinformatics. Many object classification and detection systems rely on binary classification using SVM, deciding whether an object is in a given window or not. To localize the object, sliding window approach is widely used. But this strongly increases the computational cost, because the classifier function has to be evaluated over a large set of candidate subwindows.

Suppose given a set of training samples, each marked as belonging to one of two classes, SVM can be trained from the training samples and then used to decide which class new testing samples belong to. The linear SVM can be represented as follows:

$$\begin{aligned} &< \vec{\mathbf{w}}, \vec{\mathbf{x}} > -b > 0 \Rightarrow \vec{\mathbf{x}} \in c_1, \\ &< \vec{\mathbf{w}}, \vec{\mathbf{x}} > -b \leq 0 \Rightarrow \vec{\mathbf{x}} \in c_2, \end{aligned} \tag{6.1}$$

where $\vec{\mathbf{x}} = [x_1 \ x_2 \ \ldots \ x_N]^T$ is the input sample, c_1 and c_2 are two different classes, $< \vec{\mathbf{w}}, \vec{\mathbf{x}} >= \vec{\mathbf{w}}^T \vec{\mathbf{x}}$, $.^T$ denotes matrix transposition, $< \vec{\mathbf{w}}, \vec{\mathbf{x}} > -b = 0$ is the hyperplane for determining whether $\vec{\mathbf{x}} \in c_1$ or $\vec{\mathbf{x}} \in c_2$, $\vec{\mathbf{w}}$ is the normal vector perpendicular to the hyperplane and b is a trained constant. In this chapter, we only analyze the two-class problem although the proposed approach is applicable for multi-class problems. In the followings, class c_1 is called positive class and class c_2 is called negative class. Sample $\vec{\mathbf{x}}$ is called positive sample if $\vec{\mathbf{x}} \in c_1$ and is called negative sample if $\vec{\mathbf{x}} \in c_2$.

Direct evaluation of the terms in (6.1) is called full search (FS) in this chapter. For FS, if $\vec{\mathbf{w}}^T\vec{\mathbf{x}} - b > 0$, then $\vec{\mathbf{x}} \in c_1$, otherwise, $\vec{\mathbf{x}} \in c_2$. FS requires N multiplications and N additions for each testing sample. Though accurate, FS is not preferred in applications requiring great classification speed especially when N is large. To relieve the burden of high computational time required for computing SVM, many fast algorithms are proposed [80–82] for kernel SVM. Fast algorithms for kernel SVM, however, are not applicable for linear SVM. In this chapter, we propose a transform domain SVM (TDSVM) using pruning that computes linear SVM much faster.

The chapter is organized as follows. Section 6.2 introduces a new viewpoint that considers SVM as pattern matching problem. Section 6.3 presents the TDSVM using pruning. Section 6.4 applies TDSVM for human detection. Finally, Section 6.5 presents conclusions.

6.2 Linear SVM as a Pattern Matching Problem

Pattern matching, also named as template matching, is the procedure of seeking a given pattern in given signal data. Suppose an $N_1 \times N_2$ pattern is to be sought in a given image. The pattern will be compared with candidate windows of similar size in the image. We represent the pattern as vector $\vec{\mathbf{x}}_t$ having length $N = N_1N_2$ and represent the N_{wnd} candidate windows as $\vec{\mathbf{x}}_{wnd}$ having length N, where subscripts $\cdot_t$ and $\cdot_{wnd}$ denote template and window respectively. The inner product between $\vec{\mathbf{x}}_t$ and $\vec{\mathbf{x}}_{wnd}$, i.e. $< \vec{\mathbf{x}}_t, \vec{\mathbf{x}}_{wnd} >= \vec{\mathbf{x}}_t^T\vec{\mathbf{x}}_{wnd}$, can be used for measuring the similarity between $\vec{\mathbf{x}}_t$ and $\vec{\mathbf{x}}_{wnd}$. The value $\vec{\mathbf{x}}_t^T\vec{\mathbf{x}}_{wnd}$ is also called cross-correlation in [26; 46]. The greater is $\vec{\mathbf{x}}_t^T\vec{\mathbf{x}}_{wnd}$, the more similar are $\vec{\mathbf{x}}_t$ and $\vec{\mathbf{x}}_{wnd}$. For a given threshold b, if $\vec{\mathbf{x}}_t^T\vec{\mathbf{x}}_{wnd} > b$, then the candidate $\vec{\mathbf{x}}_{wnd}$ is considered to match pattern $\vec{\mathbf{x}}_t$, otherwise, it is considered as a non-matching candidate. The condition $\vec{\mathbf{x}}_t^T\vec{\mathbf{x}}_{wnd} > b$ for matching $\vec{\mathbf{x}}_t$ and $\vec{\mathbf{x}}_{wnd}$ in pattern matching is equivalent to the condition $\vec{\mathbf{w}}^T\vec{\mathbf{x}} - b > 0$ for classifying $\vec{\mathbf{x}}$ as positive class using SVM in (6.1) if $\vec{\mathbf{w}} = \vec{\mathbf{x}}_t$ and $\vec{\mathbf{x}}_{wnd} = \vec{\mathbf{x}}$. From this viewpoint, the fast algorithms for pattern matching in [24–26; 46] can be used for SVM.

6.2.1 Fast Pattern Matching using Enhanced Bounded Correlation

The Enhanced Bounded Correlation (EBC) algorithm proposed by Mattoccia et al. [26] defines a partition of set $\{1, 2, \ldots N\}$ into N_P disjoint sub-sets $P_1, \cdots, P_{N_P}$, where

$1 \leq N_P \leq N$, $P = \{P_1, P_2, \ldots, P_{N_P}\}$, $\cup_{m=1}^{N_P} P_m = \{1, 2, \ldots N\}$, $P_u \cap P_v = \phi, \forall u \neq v, u, v \in 1, 2, \cdots, N_P$.Given P, EBC defines the *partial* L_2-norm of vector $\vec{\mathbf{x}}_t$ and $\vec{\mathbf{x}}_{wnd}$ limited to the sub-vector associated with $P_m \in P$ as:

$$||\vec{\mathbf{x}}_t||_{2,P_m} = \Big(\sum_{n \in P_m} |x_{t,n}|^2\Big)^{\frac{1}{2}}, \quad ||\vec{\mathbf{x}}_{wnd}||_{2,P_m} = \Big(\sum_{n \in P_m} |x_{wnd,n}|^2\Big)^{\frac{1}{2}}, \tag{6.2}$$

where $x_{t,n}$ is the nth element in $\vec{\mathbf{x}}_t$ and $x_{wnd,n}$ is the nth element in $\vec{\mathbf{x}}_{wnd}$.

By virtue of the *Cauchy-Schwarz inequality* applied on corresponding sub-vectors, EBC establishes the following inequality:

$$\vec{\mathbf{x}}_t^T \vec{\mathbf{x}}_{wnd} = \sum_{n=1}^{N} x_{t,n} x_{wnd,n} \leq \sum_{m=1}^{N_P} ||\vec{\mathbf{x}}_t||_{2,P_m} ||\vec{\mathbf{x}}_{wnd}||_{2,P_m} = f_{up,EBC}(\vec{\mathbf{x}}_{wnd}), \tag{6.3}$$

where $f_{up,EBC}(\vec{\mathbf{x}}_{wnd})$ is the upper bounding function for $\vec{\mathbf{x}}_t^T \vec{\mathbf{x}}_{wnd}$ in EBC. If $f_{up,EBC}(\vec{\mathbf{x}}_{wnd}) \leq b$, then $\vec{\mathbf{x}}_t^T \vec{\mathbf{x}}_{wnd} \leq b$ and $\vec{\mathbf{x}}_{wnd}$ can be safely considered as the non-matching candidate. Moreover, should (6.3) not be satisfied, rather than directly computing $\vec{\mathbf{x}}_t^T \vec{\mathbf{x}}_{wnd}$ using FS, EBC obtains successive pruning condition based on a tighter upper-bound by considering sub-vectors pairs. EBC gives the same result as FS in pattern matching.

6.3 Transform Domain Support Vector Machine using Week Upper Bound

In this section, the overall scheme of TDSVM using pruning is introduced, then details for the scheme, including TDSVM and week upper bound function, are presented.

6.3.1 The Overall Scheme of Transform Domain SVM using Pruning

Denote the set of candidates as set_{can}, which initially contains all candidate samples. The proposed TDSVM using pruning consists of two steps: *pruning step* and *FS step*. The pruning step comprises two sub-steps: in *Step a.1*, an upper bounding function $f_{up}(\vec{\mathbf{x}})$ is evaluated such that $\vec{\mathbf{w}}^T \vec{\mathbf{x}} \leq f_{up}(\vec{\mathbf{x}})$ in most cases; in *Step a.2*, if $f_{up}(\vec{\mathbf{x}}) \leq b$, then we classify $\vec{\mathbf{x}}$ as negative class and prune it from set_{can}, $f_{up}(\vec{\mathbf{x}}) \leq b$ being the pruning condition. After the pruning step, the remaining candidate windows in set_{can} undergo FS for finding out the candidates that belongs to positive class in the *FS step*.

The advantage of using $f_{up}(\vec{\mathbf{x}})$ is that it is more efficient computing $f_{up}(\vec{\mathbf{x}})$ than computing $\vec{\mathbf{w}}^T \vec{\mathbf{x}}$ and the pruning condition $f_{up}(\vec{\mathbf{x}}) \leq b$ can eliminate most negative samples. This pruning scheme lies in the fact that there are more negative testing

Initially, set_{can} contains all candidates $\vec{\mathbf{x}}$; **Pruning Step**: For $\vec{\mathbf{x}}$ in set_{can}: { **Step a.1:** Obtain $f_{up}(\vec{\mathbf{x}})$; **Step a.2:** if $f_{up}(\vec{\mathbf{x}}) \leq b$, then classify $\vec{\mathbf{x}}$ as the negative class and prune $\vec{\mathbf{x}}$ from set_{can}.} **FS Step**: The remaining candidates in set_{can} undergo FS.

Table 6.1: *The pruning scheme for TDSVM using upper bound.*

samples than positive testing samples or vise versa. For example, the number of human-absent (negative) windows is much larger than the number of human-existing (positive) windows in human detection. When most negative samples are pruned in the pruning step, there are only very small number of candidates in set_{can} for the FS step and thus computation is greatly saved.

6.3.2 Transform Domain Support Vector Machine

The transformation that projects a vector $\vec{\mathbf{x}} = [x_1\ x_2\ \ldots\ x_N]^T$ onto a linear subspace spanned by U basis vectors $\vec{\mathbf{v}}^{(N,0)}, \ldots\ \vec{\mathbf{v}}^{(N,U-1)}$ can be represented as follows:

$$\vec{\mathbf{y}}^{(U)} = \mathbf{V}^{(U\times N)}\vec{\mathbf{x}} = [\vec{\mathbf{v}}^{(N,0)}\ \ldots\ \vec{\mathbf{v}}^{(N,U-1)}]^T\vec{\mathbf{x}}, \tag{6.4}$$

where $\cdot^T$ is matrix transposition, vector $\vec{\mathbf{x}}$ of length N is called input sample, vector $\vec{\mathbf{y}}^{(U)}$ of length U is called projection value vector, the U elements in vector $\vec{\mathbf{y}}^{(U)}$ are called projection values and $\mathbf{V}^{(U\times N)}$ is a $U \times N$ matrix that contains U basis vectors $\vec{\mathbf{v}}^{(N,i)}$ of length N for $i = 0, \ldots, U-1$. The transformation is also called projection in [1; 35]. Basis vector is also called projection kernel in [1].

Denote $\vec{\mathbf{z}}^{(U)}$ as the projection of $\vec{\mathbf{w}}$ onto $\mathbf{V}^{(U\times N)}$, i.e. $\vec{\mathbf{z}}^{(U)} = \mathbf{V}^{(U\times N)}\vec{\mathbf{w}}$. When $\mathbf{V}^{(N\times N)}$ is orthogonal, i.e. $\mathbf{V}^{(N\times N)^T}\mathbf{V}^{(N\times N)} = I_N$, where I_N is an $N \times N$ identity matrix, we have:

$$\begin{aligned}
\vec{\mathbf{z}}^{(N)^T}\vec{\mathbf{y}}^{(N)} &= \left(\mathbf{V}^{(N\times N)}\vec{\mathbf{w}}\right)^T\left(\mathbf{V}^{(N\times N)}\vec{\mathbf{x}}\right) = \vec{\mathbf{w}}^T\left(\mathbf{V}^{(N\times N)^T}\mathbf{V}^{(N\times N)}\right)\vec{\mathbf{x}} = \vec{\mathbf{w}}^T\vec{\mathbf{x}}, \\
\text{where } \vec{\mathbf{z}}^{(N)} &= \mathbf{V}^{(N\times N)}\vec{\mathbf{w}} = \begin{bmatrix} \vec{\mathbf{z}}^{(U)} \\ \vec{\mathbf{z}}^{(N-U)} \end{bmatrix} = \begin{bmatrix} \mathbf{V}^{(U\times N)}\vec{\mathbf{w}} \\ \mathbf{V}^{((N-U)\times N)}\vec{\mathbf{w}} \end{bmatrix}, \\
\vec{\mathbf{y}}^{(N)} &= \mathbf{V}^{(N\times N)}\vec{\mathbf{x}} = \begin{bmatrix} \vec{\mathbf{y}}^{(U)} \\ \vec{\mathbf{y}}^{(N-U)} \end{bmatrix} = \begin{bmatrix} \mathbf{V}^{(U\times N)}\vec{\mathbf{x}} \\ \mathbf{V}^{((N-U)\times N)}\vec{\mathbf{x}} \end{bmatrix}.
\end{aligned} \tag{6.5}$$

And we have the follows from (6.5):

$$\vec{\mathbf{w}}^T\vec{\mathbf{x}} = \vec{\mathbf{z}}^{(N)^T}\vec{\mathbf{y}}^{(N)} = \vec{\mathbf{z}}^{(U)^T}\vec{\mathbf{y}}^{(U)} + \vec{\mathbf{z}}^{(N-U)^T}\vec{\mathbf{y}}^{(N-U)}. \tag{6.6}$$

Suppose we have an upper bound function $f_{up,1}$ for $\vec{\mathbf{z}}^{(N-U)^T}\vec{\mathbf{y}}^{(N-U)}$, i.e. $\vec{\mathbf{z}}^{(N-U)^T}\vec{\mathbf{y}}^{(N-U)} \leq f_{up,1}$, an upper bound function in the transform domain $f_{up,TD}$ for $\vec{\mathbf{w}}^T\vec{\mathbf{x}}$ can be obtained as follows from (6.6):

$$\vec{\mathbf{w}}^T\vec{\mathbf{x}} = \vec{\mathbf{z}}^{(U)^T}\vec{\mathbf{y}}^{(U)} + \vec{\mathbf{z}}^{(N-U)^T}\vec{\mathbf{y}}^{(N-U)} \leq \vec{\mathbf{z}}^{(U)^T}\vec{\mathbf{y}}^{(U)} + f_{up,1} = f_{up,TD}. \tag{6.7}$$

For the $\vec{\mathbf{w}}^T\vec{\mathbf{x}}$ in (6.7), $\vec{\mathbf{z}}^{(U)^T}\vec{\mathbf{y}}^{(U)}$ contains the main information, $\vec{\mathbf{z}}^{(U)^T}\vec{\mathbf{y}}^{(N-U)}$ contains the remaining information. The $\vec{\mathbf{z}}^{(U)^T}\vec{\mathbf{y}}^{(U)}$ in (6.7) is obtained by computation while the $\vec{\mathbf{z}}^{(N-U)^T}\vec{\mathbf{y}}^{(N-U)}$ is estimated using the upper bound function $f_{up,1}$.

The $\vec{\mathbf{z}}^{(U)^T}\vec{\mathbf{y}}^{(U)}$ in (6.7) can also be considered as projecting $\vec{\mathbf{w}}$ onto the subspace spanned by the basis vectors in $\mathbf{V}^{(U\times N)}$ and then compute the following inner product:

$$< \vec{\mathbf{w}}', \vec{\mathbf{x}} > = < \mathbf{V}^{(U\times N)^T}\mathbf{V}^{(U\times N)}\vec{\mathbf{w}}, \vec{\mathbf{x}} > = < \mathbf{V}^{(U\times N)}\vec{\mathbf{w}}, \mathbf{V}^{(U\times N)}\vec{\mathbf{x}} > = < \vec{\mathbf{z}}^{(U)}, \vec{\mathbf{y}}^{(U)} >, \tag{6.8}$$

where $\vec{\mathbf{w}}'$ is an approximation of $\vec{\mathbf{w}}$ using $\mathbf{V}^{(U\times N)}$.

To ensure that the framework in Table 6.1 finds the same result as FS, the upper bound function $f_{up,TD}(\vec{\mathbf{x}})$ in (6.7) should satisfy that $\vec{\mathbf{w}}^T\vec{\mathbf{x}} \leq f_{up,TD}(\vec{\mathbf{x}}), \forall\vec{\mathbf{x}}$, because if $f_{up}(\vec{\mathbf{x}}) \leq b$, then $\vec{\mathbf{w}}^T\vec{\mathbf{x}} \leq b$ and $\vec{\mathbf{x}}$ can be safely classified as negative class in the pruning step. This upper bound function can be obtained using exact inequalities such as the Cauchy-Schwarz inequality used in (6.3) for EBC. Upper bound function obtained from mathematically ensured inequalities is called strong upper bound in this chapter. However, Cauchy-Schwarz inequality is a very *strong* condition such that the equality holds if and only if $\vec{\mathbf{w}} = a\vec{\mathbf{x}}$ for any constant a, which does not happen for most SVM applications. Thus the strong upper bound tends to be too large to efficiently eliminate negative samples in the pruning step.

The example in Figure 6.1 is used for illustrating the TDSVM. Assume that there is a hyperplane to correctly classify the testing samples in this example, which corresponds to Figure 6.1. The input sample $\vec{\mathbf{x}}$ is assume to have length 2, i.e. $\vec{\mathbf{x}} = [x_1\ x_2]^T$. In TDSVM, we project the $\vec{\mathbf{w}}$ onto the subspace spanned by the basis vector $\mathbf{V}^{(U\times N)} =$

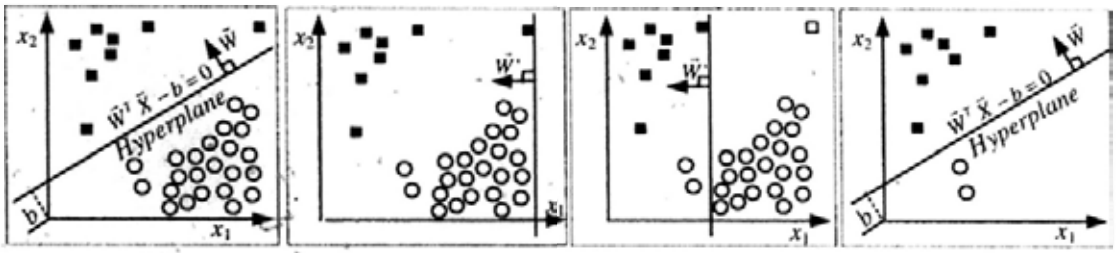


Figure 6.1: *(a) the SVM classification hyperplane* $\vec{\mathbf{w}}^T\vec{\mathbf{x}} - b = 0$*, where a testing sample is classified as positive class when it is in the upper left part of the hyperplane while the sample is classified as negative class when it is in the bottom right part of the hyperplane;* ***(b)*** *the classification hyperplane using the strong upper bound;* ***(c)*** *the classification hyperplane using the week upper bound, where samples on the right part of the plane are pruned and classified as the negative class and the white rectangle is the positive sample that is misclassified as the negative class in the pruning step;* ***(d)*** *after most samples are pruned in the pruning step, the remaining samples undergo FS using the original hyperplane. The hyperplane is a line since we have* $\vec{\mathbf{x}} = [x_1\ x_2]^T$ *in this figure. Black squares denote positive testing samples and white circles denote negative testing samples.*

[1 0] and have the follows for (6.8):

$$< \vec{\mathbf{w}}', \vec{\mathbf{x}} > = < [1\ 0]^T [1\ 0]\vec{\mathbf{w}}, \vec{\mathbf{x}} > = w_1 x_1. \tag{6.9}$$

Computing the approximated inner product $< \vec{\mathbf{w}}', \vec{\mathbf{x}} > = w_1 x_1$ is more efficient than computing $< \vec{\mathbf{w}}, \vec{\mathbf{x}} > = w_1 x_1 + w_2 x_2$. To obtain the same result as FS using the $f_{up,TD}$ in (6.7) for the pruning scheme in Table 6.1, we have to find a hyperplane that: 1) is orthogonal to $\vec{\mathbf{w}}'$; 2) will not misclassify positive samples as negative class, which corresponds to the strong upper bound. To meet these requirements, the hyperplane is not able to prune any negative samples for the example shown in Figure 6.1(b).

6.3.3 The Week Upper Bound Function

Re-investigating the scheme in Table 6.1, we find that the only mistake that makes the scheme produce different result from FS is: a positive sample $\vec{\mathbf{x}}$ is misclassified as negative class by the pruning condition $f_{up}(\vec{\mathbf{x}}) \leq b$ in *Stepa*.2. Data that are classified as negative class by FS will never be misclassified. Hence if $f_{up}(\vec{\mathbf{x}}) > b$ for all positive samples, then the output is the same as FS. Instead of finding the upper bound that has $f_{up}(\vec{\mathbf{x}}) > b$ for *all* positive samples, we can find the following week upper bound such that $f_{up}(\vec{\mathbf{x}}) > b$ for *most* positive samples, say 99% positive samples:

$$f_{up,new}(\vec{\mathbf{x}}, b) = \vec{\mathbf{z}}^{(U)^T} \vec{\mathbf{y}}^{(U)} + f_{up,w}(b), \tag{6.10}$$

where $f_{up,new}(\vec{\mathbf{x}}, b)$ is required to satisfy the inequality $f_{up,new}(\vec{\mathbf{x}}, b) > b$ for Per positive training samples $\vec{\mathbf{x}}$ and $f_{up,w}(b)$ is a constant that satisfies this requirement. For example, if $Per = 100\%$, then we choose a $f_{up,w}(b)$ such that all positive training samples will stay in set_{can} and undergo the FS step; if $Per = 99\%$, then 99% positive training samples will stay in set_{can} and undergo the FS step. Since the training samples and testing samples have similar structures, about Per positive testing samples will stay in set_{can} and undergo the FS step.

For the example in Figure 6.1, we relax the strong upper bound function and require that only very small number of positive samples are misclassified as negative class in the pruning step. Thus we have a week upper bound $f_{up,w}(b)$ and a new hyperplane $\vec{\mathbf{w}}'^T\vec{\mathbf{x}} + f_{up,w}(b) - b = 0$ in Figure 6.1(c). Using this new hyperplane, most negative samples are pruned from set_{can} with the cost that a positive sample, which is the white rectangle in Figure 6.1(c), is misclassified as negative class. Since possibility of this misclassification is small, it has little influence on the classification error rates. Finally, the FS step will classify the remaining candidates using the original hyperplane as shown Figure 6.1(d), where most negative samples are pruned by the pruning step.

6.4 Application of TDSVM to Human Detection

6.4.1 Human Detection using Histogram of Oriented Gradient

Human detection using Histogram of Oriented Gradient (HOG) as the descriptor and SVM as the classifier has been proved to be powerful [83]. And it is recently integrated into OpenCV [53]. We give a short description of this algorithm following [84]. Each detection window is divided into cells of size 8×8 pixels and each group of 2×2 cells is integrated into a block in a sliding fashion, thus blocks overlap with each other. The distances between horizontally or vertically adjacent blocks are 8 pixels. Each cell consists of a 9-bin Histogram of Oriented Gradients (HOG) and each block contains a concatenated vector of all its cells. Each block is thus represented by a L_2-norm normalized length-36 feature vector that is called HOG descriptor. Each 64×128 detection window with width 64 and height 128 is represented by 7×15 blocks, giving a total of 3780 features per detection window. Thus the input sample $\vec{\mathbf{x}}$ has length $N = 3780$. These features are then classified by a linear SVM as shown in (6.1).

Since HOG using SVM is time consuming, many researchers investigated how to

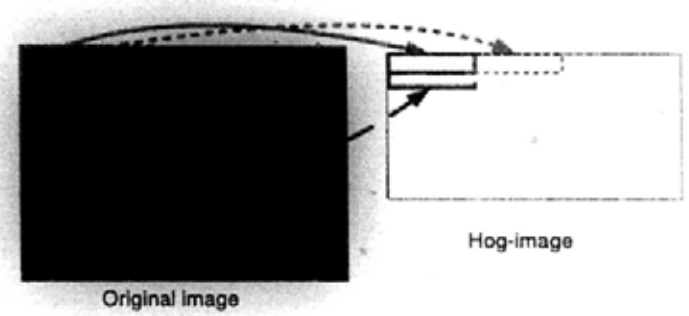


Figure 6.2: *Original image (left) and its HOG-image (right). Each* 16×16 *block on the original image is represented by a* 36×1 *descriptor on the HOG-image.*

accelerate the procedure. Wojek et al [85] implement HOG on a powerful GPU. TDSVM requires fewer operation and is also highly parallel, hence can be efficiently used for GPU. Zhu et al use the cascaded HOG for classification in [84] which is much faster than HOG. The differences between TDSVM and the cascaded HOG in [84] are: 1) the week upper bound for positive samples can be obtained from positive training samples in a few seconds while the training for cascaded HOG in [84] requires a few days; 2) the cascaded HOG is an application of boosting to HOG, TDSVM is a fast algorithm for SVM that can be used for HOG; 3) choice of week upper bound using (6.10) is different from the Adaboost training for cascaded HOG; 4) TDSVM obtains pruning condition in transform domain while the cascaded HOG obtains week classifier directly from features; 5) instead of replacing cascaded HOG, TDSVM can be used for it because the week classifiers of the cascaded HOG are linear SVMs evaluated by FS.

6.4.2 Our implementation of Transform Domain SVM for HOG

Given an image which is called *original image*, we can extract HOG descriptors from it. The HOG descriptors are arranged by a 2D array that is called *HOG-image*. A 16×16 block in the original image is represented by its 36×1 HOG descriptor in the HOG-image; relative positions for blocks in the original image corresponds to relative positions for HOG descriptors in the HOG-image. Each 64×128 detection window on the original image corresponds to a $36 \cdot 7 \times 15$ window, which is called HOG-window, on the HOG-image. For a $J_1 \times J_2$ original image, we have a $\frac{36 \cdot J_1}{8} \times \frac{J_2}{8}$ HOG-image. Detection on the original image corresponds to computing SVM classifier for each sliding HOG-window on the HOG-image. An example is shown in Figure 6.2.

The proposed transform is best explained using Kronecker product which is denoted

as $\otimes$. If $\mathbf{A}$ is a $U_1 \times Q_1$ matrix (a_{n_1,n_2}) and $\mathbf{B}$ is a $U_2 \times Q_2$ matrix (b_{m_1,m_2}), then $\mathbf{A} \otimes \mathbf{B}$ is the following $U_1U_2 \times Q_1Q_2$ matrix:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{0,0}\mathbf{B} & a_{0,1}\mathbf{B} & \cdots & a_{0,Q_1-1}\mathbf{B} \\ a_{1,0}\mathbf{B} & a_{1,1}\mathbf{B} & \cdots & a_{1,Q_1-1}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{U_1-1,0}\mathbf{B} & a_{U_1-1,1}\mathbf{B} & \cdots & a_{U_1-1,Q_1-1}\mathbf{B} \end{bmatrix}. \tag{6.11}$$

Denote the $1 \times N$ matrix with all elements equal to 0 and 1 as $\mathbf{0}_{1\times N}$ and $\mathbf{1}_{1\times N}$ respectively. Denote the 252×15 HOG-window as $X^{(15\times 252)}$, the two selected transforms are as follows:

$$\mathbf{Y}^{(5\times 126)} = \mathbf{V}_v^{(5\times 15)} \mathbf{X}^{(15\times 252)} \mathbf{V}_h^{(126\times 252)^T}, \tag{6.12}$$

$$\mathbf{Y}^{(5\times 63)} = \mathbf{V}_v^{(5\times 15)} \mathbf{X}^{(15\times 252)} \mathbf{V}_h^{(63\times 252)^T}, \tag{6.13}$$

where $\mathbf{V}_v^{(5\times 15)} = I_5 \otimes \mathbf{1}_{1\times 3}$, $\mathbf{V}_h^{(126\times 252)} = I_7 \otimes \mathbf{1}_{1\times 2} \otimes I_{18}$ in (6.12) and $\mathbf{V}_h^{(63\times 252)} = I_7 \otimes \mathbf{1}_{1\times 4} \otimes I_9$ in (6.13). The transforms for the 252×15 HOG-window represented by vector $\vec{\mathbf{x}}$ of length $N = 3780$ can be represented as follows:

$$\vec{\mathbf{y}}^{(U)} = \vec{\mathbf{y}}^{(630)} = (\mathbf{V}_h^{(126\times 252)} \otimes \mathbf{V}_v^{(5\times 15)})\vec{\mathbf{x}} = \mathbf{V}^{(630\times 3780)}\vec{\mathbf{x}} \tag{6.14}$$

$$\vec{\mathbf{y}}^{(U)} = \vec{\mathbf{y}}^{(315)} = (\mathbf{V}_h^{(63\times 252)} \otimes \mathbf{V}_v^{(5\times 15)})\vec{\mathbf{x}} = \mathbf{V}^{(315\times 3780)}\vec{\mathbf{x}} \tag{6.15}$$

6.4.3 Computational Analysis

For a $J_1 \times J_2$ original image, we have a $\frac{36}{8}J_1 \times \frac{J_2}{8}$ HOG-image. Denote the size of HOG-image as $J_3 \times J_4$, where $J_3 = \frac{36}{8}J_1$, $J_4 = \frac{J_2}{8}$. The FS requires about $3780J_3J_4/36$ additions and $3780J_3J_4/36$ multiplications. We can also use the FFT approach in [46] to compute the inner product $< \vec{\mathbf{w}}, \vec{\mathbf{x}} >$ on sliding windows. As pointed out in [46], the FFT-based approach requires about $6J_3J_4 \log_2(J_3J_4)$ additions and $6J_3J_4 \log_2(J_3J_4)$ multiplications.

The computation required by TDSVM for the transform in (6.14) is as follows: 1) the transform for obtaining the $\vec{\mathbf{y}}^{(630)}$ in (6.14) can be computed by $3J_3J_4$ additions using integral image [68]; 2) construction of the integral image requires about $2J_3J_4$ additions; 3) given $\vec{\mathbf{y}}^{(630)}$, the upper bound function $< \vec{\mathbf{z}}^{(630)}, \vec{\mathbf{y}}^{(630)} > + f_{up,w}(b)$, which is used for the pruning step in Table 6.1, can be computed by $630J_3J_4/36$ additions and

	$J_1 \times J_2$ original image	256×256	512×512	1024×1024
FS A/M	$105J_3J_4$	3,870,720	15,482,880	61,931,520
FFT A/M	$6J_3J_4log_2(J_3J_4)$	3,355,345	15,190,851	67,841,291
TDSVM A	$22.5J_3J_4 + 3780N_{can}^{(FS)}$	1,487,462	5,949,850	23,799,398
TDSVM M	$17.5J_3J_4 + 3780N_{can}^{(FS)}$	1,303,142	5,212,570	20,850,278

Table 6.2: *Number of operations required by different approaches for different image sizes, where* $J_3 = \frac{36J_1}{8}, J_4 = \frac{J_2}{8}$. *"FS A/M" denotes the number of additions or multiplications required by FS. The number of additions and multiplications required by FS is the same and thus share the same row. This is similar for FFT. For TDSVM, the rows "TDSVM A" and "TDSVM M" respectively denote the number of additions and multiplications required. We assume that 83% candidates are eliminated in the pruning step for TDSVM.*

$630J_3J_4/36$ multiplications. 4) finally, the FS step requires $3780N_{can}^{(FS)}$ additions and $3780N_{can}^{(FS)}$ multiplications for the remaining $N_{can}^{(FS)}$ candidates in set_{can}. In summary, the TDSVM using the transform in (6.14) requires $5J_3J_4 + 630J_3J_4/36 + 3780N_{can}^{(FS)}$ additions and $630J_3J_4/36 + 3780N_{can}^{(FS)}$ multiplications. The TDSVM using the transform in (6.15) can be similarly analyzed.

Table 6.2 compares the proposed TDSVM using (6.14) with FS and FFT for original images having different sizes. It can be seen that FFT requires similar operations compared with FS because the sliding HOG window has a horizontal stride of 36 pixels on the HOG-image. The speed-up of TDSVM over FS is about 3 when 83% candidates are pruned from set_{can} in the pruning step. The speed-up of TDSVM over FS in execution time or in operation is measured as the execution time or number of operations required by FS divided by that required by TDSVM in computing SVM.

6.4.4 Experimental Results on INRIA Datasets

In this experiment, the INRIA training and testing dataset provided by Dalal & Triggs online [86] for their HOG paper [83] are used as the datasets for training and testing. The experiments are implemented on a PC with 2.13GHz CPU and 3G memory using single threaded C++ on windows XP system with compiling environment VS2005. The default SVM classifier provided by Dalal & Triggs in [87] are used as the classifier for FS.

We investigate the experimental results using the following approaches:

1) FS, the original implementation in OpenCV [53], which is faster than Dalal's in [87];

2) direct TDSVM, the TDSVM without upper bound, i.e. $f_{up,w}(b)$=0 and U=630 in (6.10);

3) TDSVM$^{630,2.75}$, the TDSVM with $f_{up,w}(b) = 2.75 + b$ and $U = 630$ in (6.10);

4) TDSVM$^{315,2.75}$, the TDSVM with $f_{up,w}(b) = 2.75 + b$ and $U = 315$ in (6.10);

5) TDSVM$^{315,2.4+0.2b}$, the TDSVM with $f_{up,w}(b) = 2.4 + 0.2b + b$ and $U = 315$ in (6.10).

6) TDSVM$^{315,2.2+0.2b}$, the TDSVM with $f_{up,w}(b) = 2.20 + 0.2b + b$ and $U = 315$ in (6.10);

TDSVM$^{630,2.75}$ and TDSVM$^{315,2.75}$ are different in the number of projection values U used for $\vec{\mathbf{y}}^{(U)}$ in (6.14) and (6.15); TDSVM$^{315,2.75}$ and TDSVM$^{315,2.4+0.2b}$ have different increment of week upper bound $f_{up,w}(b)$ for the same increment of b; TDSVM$^{315,2.4+0.2b}$ and TDSVM$^{315,2.2+0.2b}$ are different in the week upper bound $f_{up,w}(b)$.

The *Detection Error Tradeoff* (DET) curves are shown in Figure 6.3 on a log-log scale, i.e. miss rate (1-Recall or $\frac{FalseNeg}{TruePos+FalseNeg}$) versus false positive per window (FPPW). Lower FPPW or miss rate corresponds to better results. According to the results, direct TDSVM performs obviously worse than FS and TDSVM using week upper bound. For the same upper bound function $f_{up,w}(b) = 2.75 + b$, the TDSVM$^{630,2.75}$ that uses 630 projection values performs better in DET than the TDSVM$^{315,2.75}$ that uses 315 projection values. For the same 315 projection values, the TDSVM$^{315,2.4+0.2b}$ that has larger upper bound performs better in DET than the TDSVM$^{315,2.2+0.2b}$ that uses smaller upper bound.

When $f_{up,w}(b) = 2.75 + b$ and $\vec{\mathbf{y}}^{(U)} = \vec{\mathbf{y}}^{(630)}$, TDSVM performs close to FS in DET. The miss rate improvement of FS over TDSVM is less than 0.1% at 10^{-4}, 10^{-3} and 10^{-2} FFPW. In this case, about 83% negative windows are eliminated and only about 0.0088 positive samples are misclassified as negative class in the pruning step. Under this setting, the speed-up in execution time of TDSVM over FS is about 2.6. When 83% candidates are eliminated at the pruning step, it is shown in Table 6.2 that the speed-up in operation of TDSVM over FS is about 3, which is close to the 2.6 in execution time. The time required for integral image and transformation has been included into the time and operation required by TDSVM in all experiments. The integral image and transformation requires about 10% the execution time of the whole TDSVM process.

When $f_{up,w}(b) = 2.4 + 0.2b + b$ and $\vec{\mathbf{y}}^{(U)} = \vec{\mathbf{y}}^{(315)}$, TDSVM also performs close to

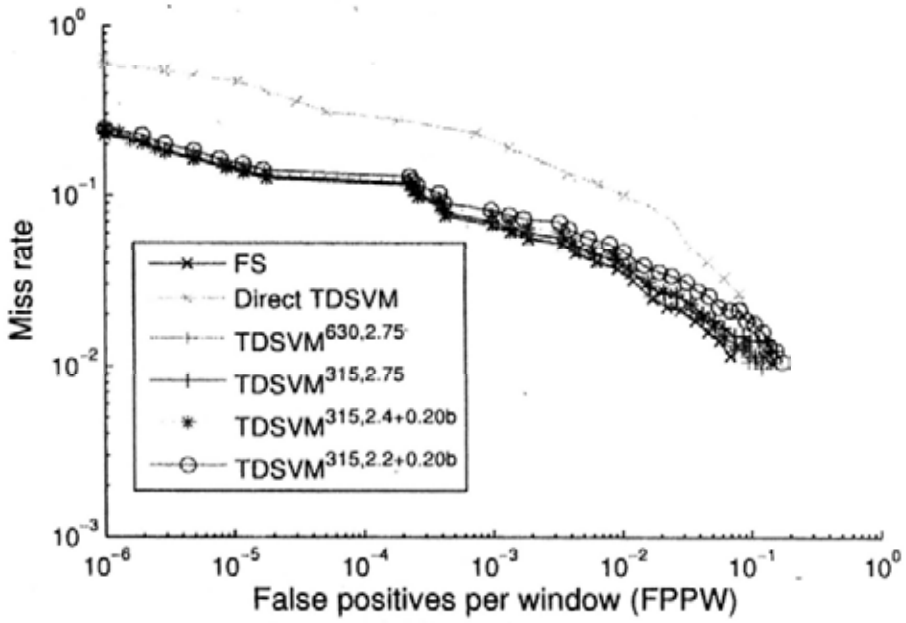


Figure 6.3: *DET curves for FS and different implementations of TDSVM.*

FS in DET. About 80% negative windows are eliminated in the pruning step at 10^{-2} FPPW. The miss rate improvement of FS over TDSVM is 0.44% at 10^{-2} FPPW, 0.62% at 10^{-3} FPPW and 0.53% at 10^{-4} FPPW. Under this setting, the speed-up in execution time of the proposed approach over FS is about 3.2 at 10^{-2} FPPW and about 4.5 at 10^{-4} FPPW. The speed-up in operation of the proposed approach over FS is about 3.5 at 10^{-2} FPPW and about 5.5 at 10^{-4} FPPW.

If we allow more degradation of miss rate at the same FPPW compared with FS, the speed-up can be further increased. By setting $f_{up,w}(b) = 2.2+0.2b+b$ and $\vec{y}^{(U)} = \vec{y}^{(315)}$, the miss rate improvement of FS over TDSVM is about 1.5% at the 10^{-2} FPPW and 10^{-4} FPPW. The speed-up in operation of the proposed approach over FS is about 4.7 at 10^{-2} FPPW and about 7 at 10^{-4} FPPW.

We also investigated the TDSVM using the strong upper bound of EBC introduced in Section 6.2.1 for the same transform in (6.12). The computation required by EBC for the upper bound function in (6.3) and the computation for TDSVM in (6.10) are the same for the same transform. We find that the strong upper bound provided by EBC in (6.3) can only eliminate less than 1% negative samples for the default SVM classifier provided by Dalal when FPPW ranges from 0 to 0.85. EBC using (6.3) is proved to be very efficient when both $\vec{w}$ and $\vec{x}$ are image data for pattern matching in [26]. However, when $\vec{x}$ is the HOG descriptor with non-negative elements and $\vec{w}$

is the SVM classifier with both positive and negative elements, EBC performs not so good because the inequality used in (6.3) is too strong in esimating the upper bound for SVM. EBC contains further seccussive pruning steps, which requires more computation in the pruning step and thus is not fair to be compared with the TDSVM using week upper bound.

6.5 Summary

This chapter proposes a fast algorithm for computing the support vector machine (SVM). Given an SVM classifier, the proposed transform domain SVM approximate the SVM using a subspace. Then an upper bound function is used for efficiently pruning large number of negative samples. When applied for human detection using Histogram of Oriented Gradients (HOG), the proposed algorithm accelerates the computation of SVM. With time saved for computing SVM, more sophisticated descriptors can be used for increasing the classification performance at the same computational time.

Chapter 7

Conclusions

Many image processing and computer vision applications require comparing a given pattern with all candidate windows. In many such applications, however, a main problem is the high computational time required by pattern matching. This thesis aims at improving the computational efficiency in pattern matching.

7.1 Contributions of the Thesis

In Chapter 2, a fast algorithm is proposed for Walsh Hadamard Transform on sliding windows. This algorithm can be used to implement pattern matching efficiently [33]. The computational requirement of the proposed algorithm is about 1.5 additions per projection vector per sample. The proposed algorithm achieves its high efficiency in the computation of order-N WHT by using order-4 and order-$N/4$ WHT.

After developing the fast Walsh Hadamard Transform (WHT) algorithm in Chapter 2, Chapter 3 proposes an analysis and comparison of state-of-the-art algorithms for full search equivalent pattern matching [88]. Our intention is that the datasets and tests used in our evaluation will be a benchmark for testing future pattern matching algorithms, and that the analysis concerning the state-of-the-art algorithms could inspire new fast algorithms. We also propose extensions of the evaluated algorithms and show that they outperform the original algorithms.

The Gray-Code Kernels (GCK) family which has WHT on sliding windows as a member is a family of kernels that can perform image analysis efficiently using a fast algorithm such as the GCK algorithm. The GCK has been successfully used for pattern matching. In Chapter 4, we develop a new family of transforms called the Kronecker-Hadamard Transform (KHT) of which the GCK family is a subset. Thus, KHT provides more choices of transforms for representing images. Then we propose a new fast

algorithm that is more efficient than the GCK algorithm. The proposed fast KHT algorithm requires 4 additions per pixel for computing 3 basis vectors independent of transform size and dimension. All KHTs can be computed efficiently using the fast KHT algorithm. Based on the KHT, we then propose the segmented KHT (SegKHT). By segmenting input data into L_s parts, the SegKHT requires only 4 additions per pixel for computing $3L_s$ basis vectors. Experimental results show that the proposed algorithm can significantly accelerate the full-search equivalent pattern matching process and outperforms state-of-the-art methods [89]. KHT and SegKHT were described in the context of transform domain pattern matching. However, pattern matching is only an application example. The properties of KHT and SegKHT make them attractive for many applications which require transformation on sliding windows such as image based rendering, image compression, super resolution, object detection, texture synthesis, block matching in motion estimation, image denoising, action recognition and wide baseline image matching.

After that, strip sum is proposed in Chapter 5. The sum of pixels in a rectangle, which is called rectangle sum, can be computed by one addition using the strip sum. 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 [42; 48; 77]. Then we propose to use the OHT for sliding transform. Existing fast algorithms for computing transformation on sliding windows [1; 32; 33] require $O(u)$ additions for projecting input data having size N onto u 2D basis. The OHT can project input data onto u 2D basis 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 [78].

Finally, transform domain SVM (TDSVM) using pruning is proposed in chapter 6. Given an SVM classifier, the proposed transform domain SVM approximate the SVM using a subspace. Then an upper bound function is used for efficiently pruning large number of negative samples. When applied for human detection using Histogram of Oriented Gradients (HOG), the proposed algorithm accelerates the computation of SVM. With time saved for computing SVM, more sophisticated descriptors can be used

for increasing the classification performance at the same computational time.

7.2 Future Work

Some suggestions for future work are listed below:

1. Apply the proposed fast algorithms to more applications. Transform domain pattern matching has found application in block matching in motion estimation for video coding [2; 9], tracking [13; 14], feature point based image matching [15], texture synthesis [49] and augmented reality [50]. We developed many fast algorithms for transform domain pattern matching and applied the algorithm to computing SVM. In the future, we can use the proposed algorithm for more applications.

2. Apply the proposed fast algorithms for pattern matching that is invariant to rotation and affine distortion. Currently, the proposed algorithms are applied when SSD and SAD are used as the measure for matching. It is clearly shown in [1] that WHT is able to deal with scale, illumination and Gaussian noise disturbances. Our experimental results in [89] show that WHT and KHT are able to deal with image blur and JPEG compression. Since WHT is used for normalized cross correlation (NCC) in [90], our proposed algorithms are naturally applicable for measures using convolution, e.g. NCC. The approach in [15] has extended WHT for dealing with rotation and affine transform. Research on using our proposed algorithms for rotation and affine distortion is an interesting research topic.

3. Design more efficient feature extraction algorithms. We have designed fast algorithm for obtaining the sum of pixels in a rectangle and for computing Haar-like features. The design of efficient feature extraction algorithms is a future work.

4. Combine the algorithm with hardware requirement and design hardware specific algorithms. Our proposed algorithm is analyzed theoretically and implemented on PC. However, the computational efficiency of the proposed algorithm to specific hardware environment is unclear and can be studied further.

5. Find the subset of KHT that is more efficient that WHT in compacting energy

from input data. Currently, we find that the Segmented KHT is more efficient than GCK in computational efficiency with some loss of energy packing ability. However, there are applications where computational efficiency is less important than energy packing ability. Thus the subset of KHT that is more efficient in compacting energy from input data can be used for these applications.

Bibliography

[1] Y. Hel-Or and H. Hel-Or, "Real time pattern matching using projection kernels," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 9, pp. 1430–1445, Sept. 2005. xvii, 3, 6, 7, 8, 11, 12, 13, 14, 29, 30, 31, 33, 36, 37, 41, 42, 44, 45, 59, 60, 65, 88, 89, 93, 107, 118, 119, 122, 124, 129, 140, 141

[2] Y. Moshe and H. Hel-Or, "Video block motion estimation based on Gray-code kernels," in *IEEE Trans. Image Process.*, vol. 18, no. 10, Oct. 2009, pp. 2243–2254. xvii, 1, 8, 22, 37, 45, 77, 89, 119, 141

[3] M. S. Aksoy, O. Torkul, and I. H. Cedimoglu, "An industrial visual inspection system that uses inductive learning," *Journal of Intelligent Manufacturing*, vol. 15, no. 4, pp. 569–574, 2004. 1

[4] A. Fitzgibbon, Y. Wexler, and A. Zisserman, "Image-based rendering using image-based priors," in *ICCV*, vol. 2, 2003, pp. 1176–1183. 1

[5] T. Luczak and W. Szpankowski, "A suboptimal lossy data compression based on approximate pattern matching," *IEEE Trans. Information Theory*, vol. 43, no. 5, pp. 1439–1451, 1997. 1

[6] R. M. Dufour, E. L. Miller, and N. P. Galatsanos, "Template matching based object recognition with unknown geometric parameters," *IEEE Trans. Image Process.*, vol. 11, no. 12, pp. 1385–1396, Dec. 2002. 1

[7] W. Freeman, T. Jones, and E. Pasztor, "Example-based super-resolution," *IEEE Computer Graphics and Applications*, vol. 22, no. 2, pp. 56–65, Mar./Apr 2002. 1

[8] A. Efros and T. Leung, "Texture synthesis by non-parametric sampling," in *ICCV*, Sept. 1999, pp. 1033–1038. 1

[9] C. Mak, C. Fong, and W. Cham, "Fast motion estimation for H.264/AVC in Walsh Hadamard domain," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no. 5, pp. 735–745, Jun. 2008. 1, 8, 37, 141

[10] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," in *CVPR*, vol. 2, Jun. 2005, pp. 60– 65. 1

[11] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *IEEE Trans. Image Process.*, vol. 16, no. 8, pp. 2080–2095, Aug. 2007. 1

[12] R. Zhang, W. Ouyang, and W. Cham, "Image deblocking using dual adaptive fir wiener filter in the DCT transform domain," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP)*, Taiwan, April 19-24 2009, pp. 1181–1184. 1

[13] Y. Alon, A. Ferencz, and A. Shashua, "Off-road path following using region classification and geometric projection constraints," in *CVPR*, vol. 1, Jun. 2006, pp. 689–696. 1, 8, 141

[14] Y. Shina, J. S. Jua, and E. Y. Kim, "Welfare interface implementation using multiple facial features tracking for the disabled people," *Pattern Recognition Letters*, vol. 29, no. 13, pp. 1784–1796, Oct. 2008. 1, 8, 141

[15] Q. Wang and S. You, "Real-time image matching based on multiple view kernel projection," in *CVPR*, 2007. 1, 8, 61, 141

[16] S. Wei and S. Lai, "Robust and efficient image alignment based on relative gradient matching," *IEEE Transa. Image Processing*, vol. 15, no. 10, pp. 2936–2943, Oct. 2006. 1

[17] X. Wu, "Template-based action recognition: Classifying hockey players movement," Master's thesis, The University of British Columbia, 2005. 1

[18] D. Lowe, "Distinctive image features from scale-invarian keypoints," *Int'l J. Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004. 2

[19] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 10, pp. 1615–1630, Oct. 2005. 2

[20] H. Bay, T. Tuytelaars, and L. Gool, "Surf; Speeded up robust features," in *ECCV*, vol. 1, 2006, pp. 404–417. 2, 107

[21] V. Lepetit and P. Fua, "Keypoint recognition using randomized trees," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 9, pp. 1465–1479, Sept. 2006. 2

[22] J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid, "Local features and kernels for classification of texture and object categories: A comprehensive study," *Int'l J. Computer Vision*, vol. 73, no. 2, pp. 213–238, 2007. 2

[23] O. Pele and M. Werman, "Robust real-time pattern matching using bayesian sequential hypothesis testing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 8, pp. 1427–1443, Aug. 2008. 2

[24] L. D. Stefano and S. Mattoccia, "Fast template matching using bounded partial correlation," *Mach. Vis. Appl.*, vol. 13, pp. 213–221, 2003. 2, 127

[25] ——, "A sufficient condition based the cauchyschwarz inequality for efficient template matching," in *Proc. IEEE Int. Conf. Image Processing*, vol. 1, Sep. 2003, pp. 269–272. 2, 127

[26] S. Mattoccia, F. Tombari, and L. D. Stefano, "Fast full-search equivalent template matching by enhanced bounded correlation," *IEEE Trans. Image Process.*, vol. 17, no. 4, pp. 528–538, Apr. 2008. 2, 3, 107, 127, 137

[27] B. Girod, *Whats Wrong with Mean-Squared Error?* MIT Press, 1993., ch. 15. 3

[28] S. Santini and R. Jain, "Similarity measures," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, no. 9, pp. 871–883, Sept. 1999. 3

[29] A. Ahumada, "Computational image quality metrics: A review," in *Proc. Soc. Information Display Intl Symp.*, vol. 24, 1998, pp. 305–308. 3

[30] M. G. Alkhansari, "A fast globally optimal algorithm for template matching using low-resolution pruning," *IEEE Trans. Image Process.*, vol. 10, no. 4, pp. 526–533, Apr 2001. 3, 29, 30, 33, 34, 36, 39, 40, 45, 82

[31] F. Tombari, S. Mattoccia, and L. D. Stefano, "Full search-equivalent pattern matching with incremental dissimilarity approximations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 1, pp. 129–141, Jan. 2009. 3, 4, 29, 30, 33, 37, 45, 82, 88, 89, 90, 92, 107, 118, 119, 120

[32] G. Ben-Artz, H. Hel-Or, and Y. Hel-Or, "The Gray-code filter kernels," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 3, pp. 382–393, Mar. 2007. 3, 6, 8, 11, 13, 14, 15, 17, 20, 29, 30, 33, 35, 37, 45, 64, 68, 69, 73, 75, 76, 84, 88, 107, 118, 124, 140

[33] W. Ouyang and W. Cham, "Fast algorithm for Walsh Hadamard transform on sliding windows," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 1, pp. 165–171, Jan. 2010. 3, 29, 30, 33, 37, 46, 62, 63, 64, 74, 81, 107, 117, 124, 125, 139, 140

[34] G. J. VanderBrug and A. Rosenfeld, "Two-stage template matching," *IEEE Trans. Comput.*, vol. C-26, no. 4, pp. 384–393, Apr. 1977. 3

[35] M. Ben-Yehuda, L. Cadany, and H. Hel-Or, "Irregular pattern matching using projections," in *Proc. 12th Int'l Conf. Image Processing (ICIP)*, vol. 2, 2005, pp. 834–837. 3, 6, 8, 61, 82, 129

[36] A. Goshtasby, *2-D and 3-D Image Registration for Medical, Remote Sensing and Industrial Applications.* New York: Wiley, 2005. 3

[37] B. Zitova and J. Flusser, "Image registration methods:a survey," *Image Vis. Comput.*, vol. 21, no. 11, pp. 977–1000, 2003. 3

[38] W. Krattenthaler, K. Mayer, and M. Zeiler, "Point correlation: A reduced-cost template matching technique," in *Proc. 1st IEEE Int. Conf. Image Processing*, vol. 1, Austin, TX, 1994, p. 208212. 3

[39] K. Briechle and U. D. Hanebeck, "Template matching using fast normalized cross correlation," in *Proc. SPIE AeroSense Symp.*, vol. 4387, no. 1. SPIE, 2001, pp. 95–102. [Online]. Available: http://link.aip.org/link/?PSI/4387/95/1 3

[40] P. S. Heckbert, "Filtering by repeated integration," in *Proc. SIG-GRAPH*, 1986, pp. 315–321. 3

[41] P. Simard, L. Bottou, P. Haffner, and Y. L. Cun, "Boxlets: A fast convolution algorithm for signal processing and neural networks," *Adv. Neural Inf. Process. Syst.*, vol. 11, pp. 571–577, 1999. 3, 107

[42] F. Tang, R. Crabb, and H. Tao, "Representing images using nonorthogonal Haar-like bases," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 12, pp. 2120–2134, Dec. 2007. 3, 124, 140

[43] C. D. Bei and R. M. Gray, "An improvement of the minimum distortion encoding algorithm for vector quantization," *IEEE Trans. Commun.*, vol. COM-33, no. 10, pp. 1132–1133, Oct. 1985. 3

[44] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Trans. Image Process.*, vol. 4, no. 1, pp. 105–107, Jan. 1995. 3, 29

[45] H. S. Wang and R. M. Mersereau, "Fast algorithms for the estimation of motion vectors," *IEEE Trans. Image Process.*, vol. 8, no. 3, p. 435439, Mar. 1999. 3, 29

[46] J. Lewis, "Fast template matching," in *Vision Interface 95*, Quebec City, Canada, May 15-19 1995, pp. 120–123. 4, 127, 134

[47] M. J. McDonnell, "Box-filtering techniques," *Comput. Graph. Image Process.*, vol. 17, pp. 65–70, 1981. 4, 41, 74, 90, 112

[48] P. Viola and M. Jones, "Robust real-time face detection," *Int'l J. Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004. 4, 63, 74, 87, 124, 140

[49] Y. Hel-Or, T. Malzbender, and D. Gelb, "Synthesis and rendering of 3d textures," in *Texture 2003 Workshop accomp. ICCV 2003,*, 2003, pp. 53–58. 8, 141

[50] Q. Wang, J. Mooser, S. You, and U. Neumann, "Augmented exhibitions using natural features," *Int'l. J. Virtual Reality*, vol. 7, no. 4, pp. 1–8, 2008. 8, 141

[51] G. Ben-Artzi, "Gray-code filter kernels (GCK)-fast convolution kernels," Master's thesis, Bar-Ilan Univ., , Ramat-Gan, Israel, 2004. 8, 65, 69, 118

[52] W. Cham and R. Clarke, "Dyadic symmetry and Walsh matrices," *IEE Proceedings, Pt.F.*, vol. 134, no. 2, pp. 141–145, April 1987. 25, 65

[53] ;, "http://sourceforge.net/projects/opencvlibrary." 29, 30, 33, 88, 114, 132, 135

[54] ——, "http://people.csail.mit.edu/torralba/images." 44, 89, 119

[55] ——, "www.data-compression.info/corpora/lukascorpus." 44, 89, 119

[56] ——, "http://zulu.ssc.nasa.gov/mrsid." 44, 89, 119

[57] ——, "www.faculty.idc.ac.il/toky/software/software.htm." 45, 89, 119

[58] Y. Geadah and M. Corinthios, "Natural, dyadic, and sequency order algorithms and processors for the Walsh-Hadamard transform," *IEEE Trans. Computers*, vol. C-26, no. 5, pp. 435 – 442, May 1977. 65

[59] H. Schweitzer, R. F. Anderson, and R. A. Deng, "A dual bound algorithm for very fast and exact template-matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, Accepted. 65, 116

[60] C.-H. Huang, J. R. Johnson, and R. Johnson, "A tensor product formulation of strassen's matrix multiplication algorithm," *Appl. Math Letters*, vol. 3, no. 3, pp. 104–108, 1990. 66

[61] E. Feig and S. Winograd, "Fast algorithms for the discrete cosine transform," *IEEE Trans. Signal Processing*, vol. 40, no. 9, pp. 2174–2193, Sept. 1992. 66

[62] A. Graham, *Kronecker Products and Matrix Calculus: With Applications.* 605 THIRD AVE., NEW YORK, NY 10158: JOHN WILEY & SONS, INC.,, 1982. 66

[63] A. Soman and P. Vaidyanathan, "Coding gain in paraunitary analysis/synthesis systems," *IEEE Trans. Signal Processing*, vol. 41, no. 5, pp. 1824–1835, May 1993. 85

[64] H. Malvar and D. Staelin, "The lot: Transform coding without blocking effects," *IEEE Transactions on Acoustics Speech and Signal processing*, vol. 37, no. 4, pp. 553–559, April 1989. 85

[65] A. Jain, "Advances in mathematical models for image processing," *Proceedings of the IEEE*, vol. 69, no. 5, pp. 502–528, May 1981. 85

[66] H. Kitajima, "Energy packing efficiency of the hadamard transform," *IEEE Trans. Communications*, vol. 24, pp. 1256–1258, 1976. 87

[67] P. Yip and K. Rao, "Energy packing efficiency for the generalised discrete transform," *IEEE Trans. Communications*, vol. 26, pp. 1256–1262, 1978. 87

[68] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *CVPR*, 2001, pp. I:511–I:518. 107, 108, 112, 134

[69] F. Crow, "Summed-area tables for texture mapping," in *Proc.11th Ann. Conf. Computer Graphics and Interactive Techniques*, 1984, pp. 207–212. 107, 108

[70] S. Maji, A. C. Berg, and J. Malik, "Classification using intersection kernel support vector machines is efficient," in *IEEE Conf. CVPR*, 2008. 107

[71] F. Porikli, "Integral histogram: A fast way to extract histograms in cartesian spaces," in *IEEE Int. Conf. Computer Vision and Pattern Recognition (CVPR)*, vol. 1, 2005, pp. 829–836. 107

[72] P. Viola, M. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance," in *ICCV*, 2003, pp. II:734–II:741. 107

[73] O. Tuzel, F. Porikli, and P. Meer, "Pedestrian detection via classification on riemannian manifolds," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 10, pp. 1713–1727, Oct. 2008. 107

[74] 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*, 2006, pp. 1491–1498. 107

[75] C. Lampert, M. Blaschko, and T. Hofmann, "Beyond sliding windows: Object localization by efficient subwindow search," in *CVPR*, 2008, pp. 1–8. 107

[76] H. Schweitzer, J. W. Bell, and F. Wu, "Very fast template matching," in *ECCV*, 2002, pp. 358–372. 107, 108

[77] 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*, 2003, pp. 297–304. 107, 108, 111, 124, 140

[78] W. Ouyang, R. Zhang, and W.-K. Cham, "Fast pattern matching using orthogonal Haar transform," in *Proc. IEEE conf. Comper vision and Pattern recognition (CVPR)*, 2010. 109, 140

[79] B. Fino, "Relations between haar and walsh/hadamard transforms," *Proceedings of the IEEE*, vol. 60, no. 5, pp. 647–648, 1972. 124

[80] S. Keerthi, O. Chapelle, and D. DeCoste, "Building support vector machines with reduced classifier complexity," *The Journal of Machine Learning Research*, vol. 7, pp. 1493–1515, 2006. 127

[81] C. J. C. Burges and B. Scholkopf, "Improving the accuracy and speed of support vector learning machines," in *Proc. the 9th NIPS Conference*, 1997, pp. 375–381. 127

[82] B. Scholkopf, C. J. C. B. S. Mika, P. Knirsch, K. R. Muller, G. Raetsch, and A. J. Smola, "Input space vs. feature space in kernel-based methods," *IEEE Trans. Neural Networks*, vol. 10, pp. 1000–1017, 1999. 127

[83] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2005. 132, 135

[84] Q. Zhu, M. Yeh, K. Cheng, and S. Avidan, "Fast human detection using a cascade of histograms of oriented gradients," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition, CVPR*, 2006. 132, 133

[85] C. Wojek, G. Dorko, A. Schulz, and B. Schiele, "Sliding-windows for rapid object class localization: A parallel technique," in *In Proc. 30th DAGM sym. Pattern Recognition*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 71–81. 133

[86] ;, "http://pascal.inrialpes.fr/data/human/." 135

[87] N. Dalal, "http://www.navneetdalal.com/software." 135

[88] W. Ouyang, F. Tombari, S. Mattoccia, L. D. Stefano, and W.-K. Cham, "Performance evaluation of full search equivalent pattern matching algorithms," *IEEE Trans. Pattern Anal. Mach. Intell.*, Under review. 139

[89] W. Ouyang, R. Zhang, and W. Cham, "The Kronecker-Hadamard transform for fast pattern matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, Under review. 140, 141

[90] W. Pan, S. Wei, and S. Lai, "Efficient NCC-based image matching in Walsh-Hadamard domain," in *ECCV*, D. Forsyth, P. Torr, and A. Zisserman, Eds., 2008. 141