ABSTRACT

Title of dissertation: EFFICIENT EVALUATION OF GAUSSIAN SUMS WITH APPLICATIONS IN VISION AND LEARNING

Changjiang Yang, Doctor of Philosophy, 2005

Dissertation directed by: Professor Larry Davis and Professor Ramani Duraiswami
Department of Computer Science

Evaluating sums of multivariate Gaussians is a common computational task in image processing, computer vision and learning, including in the general and powerful technique of kernel density estimation (KDE). The quadratic computational complexity of KDE is a significant barrier to practical applications. The fast Gauss transform (FGT) has successfully accelerated the kernel density estimation to linear time for low-dimensional problems. Unfortunately, direct extension of the FGT to higher-dimensional problems results in an algorithm whose cost grows exponentially with dimension, making it impractical for dimensions above three. We develop an improved fast Gauss transform (IFGT) in higher dimensions. A new multivariate expansion scheme and an adaptive space subdivision technique dramatically improve the performance. The IFGT has been applied to the mean shift algorithm achieving linear computational complexity, and been applied to applications such as image segmentation, and visual tracking.

Another issue when applying kernel methods to vision problems is that the statistical similarity measures commonly used were meant for low-dimensional prob-

lems. In particular similarity measures such as the Bhattacharya coefficient or the K-L distance were computed using histograms. When these measures are used for high-dimensional problems it is found that they are either unstable or not discriminative, and further are expensive to compute. We proposed a novel simple symmetric similarity function between kernel-density estimates of the model and target distributions. The mean-shift algorithm is used to track objects by iteratively maximizing this similarity function. To alleviate the quadratic complexity of the density estimation, we employ Gaussian kernels and the IFGT, which leads to very efficient and robust nonparametric tracking algorithms. The proposed algorithms are tested with several image sequences and shown to achieve robust real-time tracking performance.

The amount of computation required for the recently popular kernel machines used in learning grows with N training samples at least as $O(N^2)$. Such a computational complexity is significant even for moderate size problems and is prohibitive for large datasets. We present an approximation technique based on the IFGT to reduce the computation into $O(N)$. We also give error bound for the approximation, and show experimental results on the UCI datasets without significant decreasing in the accuracy.

EFFICIENT EVALUATION OF GAUSSIAN SUMS WITH APPLICATIONS IN VISION AND LEARNING

by

Changjiang Yang

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2005

Advisory Committee:

Professor Larry Davis, Chair/Advisor
Professor Ramani Duraiswami, Co-Advisor
Professor David Jacobs
Professor Dianne P. O'Leary
Professor Jian-Guo Liu

DEDICATION

To my parents, for all their sacrifices.

誰言寸草心，報得三春暉

ACKNOWLEDGMENTS

It is impossible to fully and thoroughly express my gratitude and appreciation to those who guided and helped me during the past five years of my graduate training. It is their guidance, encouragement and support that made this thesis possible and my graduate study an experience I will cherish forever.

First I would like to thank my advisors, Professor Larry Davis and Ramani Duraiswami, for giving me an invaluable opportunity to work on challenging and extremely interesting projects over the past years, guiding and encouraging me throughout this research.

Professor Larry Davis has always made himself available for advice and there has never been a single occasion when I've knocked on his door and he hasn't given me time. He oversaw my work and always gave me right direction to follow. Professor Ramani Duraiswami contributed enormously toward my research and guided me closely through my graduate study. He gave me the opportunity to explore my ideas, while ensuring that I was following a reasonable path. Sometimes even when I showed him some naive ideas, he still patiently listened to me and pointed out the problems concisely and exactly. He also helped me a lot on my language skills. Every time I made a mistake on pronunciation or grammar, he would carefully nudge me. It has been a pleasure to work with and learn from them.

I am most grateful to Professor David Jacobs, Professor Dianne P. O'Leary

and Professor Jian-Guo Liu for serving on my defense committee. Their courses greatly enriched my understanding on my research. I also thank them for their invaluable suggestions, comments and advice. Professor O'Leary's elegant class notes and wonderful classes provided me a perfect resource for optimization and numerical analysis. She also gave me invaluable suggestions and encouragements on my work. Professor Jacobs taught me the helpful knowledge on pattern recognition and image segmentation. Professor Liu's course greatly extended my understanding on PDE. I also want to thank Professor Hanan Samet and Professor David Mount for their helpful suggestions and discussions.

My colleagues at the Computer Vision Laboratory and the Perceptual Interfaces and Reality Laboratory have enriched my graduate life in many ways and deserve a special mention. I would like to thank Chiraz, Ahmed, Thanarat, Anurag, Zhiyun, Ali, Harsh, Kyungnam, Kyongil, Sernam, Shiv, Vinay, Son, Bohyung, Kexue, Zhihui and many others. It is impossible to mention all, and I apologize to those I inadvertently left out. I also thank the members of the Vision and Learning Reading Group. We discussed papers weekly and I have learnt a lot from you.

I owe my deepest thanks to my family – my mother, father and my wife who have always supported me, understood me and encouraged me through all the ups and downs during the graduate study. Words cannot express the gratitude I owe them.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

The ultimate goal of computer vision is to build a machine which can see the world as human beings see it. Pictures or sequences of pictures are used to extract descriptions of the world scenes. Computer vision has a wide range of applications, such as robot navigation, image registration, image retrieval, human computer interaction, medical image processing, and virtual reality [4, 82, 65, 116, 52, 46, 63].

During the past decades, computer vision has achieved great success both theoretically and practically. The field has been established since the 1960s. During the early stage of computer vision, it is almost equivalent to the image processing. It is only recently that many vision systems and real-world applications for various tasks have been developed. This is partly because of the faster computers and better imaging systems in lower price. A more important reason is the rapid growth and active research in this field. Another reason is the interdisciplinary studies with other fields such as artificial intelligence, machine learning, psychology, computer graphics, and medical image processing. Generally, this field is fast-growing and promising.

In traditional computer vision, the processing is on the image pixel level. The trend today is towards much richer data sources such as continuous video streams,

or massive DNA microarray datasets. The systems to process these large datasets should consider the information higher than the pixel level. Usually these real-world problems in vision and learning are complicated, high-dimensional, non-linear and large-scale. To solve these problems, efficient algorithms are indispensable. Therefore scientific computing has played an important role in vision and learning, and it will continuously be more important.

One of the important solutions to these high-dimensional real-world problems is the kernel method [121, 117, 118, 104]. In kernel method, a higher-dimensional Reproducing Kernel Hilbert Space (RKHS) [121] is employed to learn the knowledge from the complicated data. In the early years of research in learning and pattern recognition, a dominant principle on the high-dimensional data analysis is the *curse of dimensionality* which is coined by Bellman in [8]. Such phenomenon of the difficulty in dealing with high-dimensional spaces is well known in many areas, such as pattern recognition, optimization, function approximation, and numerical integration [33]. However, another principle on the high-dimensional data analysis, the *blessings of dimensionality*, is less widely recognized [31]. It means that the lower-dimensional spaces are lifted up into higher-dimensional spaces. The complicated relations in lower dimensions become much simpler in the higher dimensions. In statistics, increase in dimensionality is often helpful to statistical analysis. The high dimensionality increases the regularity of the data, where many statistical tools can be utilized. Such a phenomenon can be partly stated as the "concentration of measure" [78].

The research on kernel method or kernel machine is currently very active and

quite comprehensive because of the importance of the high dimensionality. Many books and conferences are dedicated to the kernel method. The focus of current research is on the performance or accuracy of the algorithms. Much little attention is paid to the another important issue of the kernel method: efficiency. However, the trend today is that data will surely become a dominative factor in this new century. The success of Google and the growth in bioinformatics have already confirmed the importance of the data, especially massive data. With the advances in data collection, faster computers, and the internet techniques, more and more data are available for processing. The demand for efficient processing techniques becomes more and more urgent.

Unfortunately, the kernel methods require a kernel density estimation. A direct evaluation is computationally of quadratic order, which seriously impairs the applicability and scalability of kernel methods on large datasets. Many efforts have been focussed on this issue. Notably the nearest neighbor searching is widely applied in the statistical applications in fields, such as machine learning, pattern recognition, information retrieval, and computer vision. The nearest neighbor searching includes two basic techniques: the tree-based methods [88] and the locality-sensitive hashing [67]. Nearest neighbor searching have achieved success in many fields, such as data retrieval, etc. But for kernel smoothing or kernel density estimation, it is not suitable for applying the nearest neighbor searching directly, because the number of neighboring data points is too large for large or even mid-size bandwidth. It is not enough for the data structures to solve the issue by themselves. We proposed an algorithm which exploits both the geometric data structures and the numerical

approximation to solve the kernel density estimates more efficiently and effectively.

Object tracking is a very active and important research area in computer vision. It has many applications and also is a preliminary step for many other more advanced or more sophisticated techniques. There are many successful methods for object tracking, among which two algorithms stand out: the mean shift tracking algorithm [24] and the particle filter based tracking algorithm [71]. The mean shift tracking algorithm is a deterministic tracker, which is robust and efficient for single object tracking. But the original mean shift tracking algorithm has no built-in mechanism for scale tracking or even more general motion models. It uses Bhattacharyya distance which does not scale well into higher dimensions. The particle filter based tracking algorithm employs multiple hypotheses about the tracked objects and is more suitable for multiple object tracking. The price is that it is not easy to achieve real-time performance, especially when many objects are presented. In this thesis, we propose a new similarity measure in the joint feature-spatial spaces, which can deal with more general motion models. For multiple object tracking, we propose a tacking algorithm based on the particle filter framework using the integral images and the edge orientation histogram.

1.1 Summary of Contributions

The main contribution of this thesis is an efficient algorithm for evaluation of the sums of Gaussians. We also applied this algorithm to many applications in vision and learning. There are three major contributions in this thesis.

1.1.1 Improved Fast Gauss Transform

Statistical applications in fields, such as machine learning, pattern recognition, information retrieval, and computer vision, often involve large-scale problems. Many algorithms for statistical inference have an $O(N^2)$ time and memory dependence on the size of the dataset N, which is too expensive for large-scale problems. The N^2 cost arises whenever kernel density estimate is computed. The list of learning problems includes Gaussian processes, radial basis networks, Support Vector Machines(SVMs), belief propagation, particle smoothing, population Monte Carlo methods, multidimensional scaling, spectral clustering, kernel methods for classification, regression, and semi-supervised learning. The matrix in these problems is commonly sparse, which allows low-rank approximation. The Fast Multipole Methods (FMM) [55], arising in the rapid calculation of physical N-body forces in gravitation/molecular dynamics, allows for approximate but arbitrarily accurate computation of matrix-vector products. In 1991, Greengard and Strain applied FMM ideas to the weighted sums of Gaussians and introduced the fast Gauss transform [57]. It reduces the complexity to linear order in low dimensional spaces, but does not scale well in higher dimensions. In fact its complexity rises exponentially with dimension. To solve these problems, we developed the improved fast Gauss transform [133] which speeds up the computation to linear order even in higher dimensions. There are two innovations in the improved fast Gauss transform. One is the multivariate Taylor expansion of the Gaussian function; and one is the k-center algorithm from the computational geometry, which partitions the data points uni-

formly and compactly. The improved fast Gauss has attracted attention from both industry and academia. We have successfully applied it to vision and learning applications, such as image segmentation, spectral clustering, kernel machines, and object tracking [133, 129, 130].

1.1.2 Robust and Efficient Similarity Measure

Many methods in vision or learning require the evaluation of the similarity measures between two probability distributions. The commonly used similarity measures include Kullback-Leibler distance and Bhattacharyya distance [37, 24]. They are limited to one or two feature dimensions, due to the difficulty in estimating the entropy of the high-dimensional features. We proposed a similarity measure which is inspired by the success of the SVMs in high dimensions. The similarity measure is the sum of all pair-wise kernelized distances between two distributions. The IFGT has been applied to reduce the computational complexity of the similarity measure from quadratic order to linear order. The robustness and efficiency of the similarity measure has been confirmed by simulations and real-world object tracking application [130].

1.1.3 Visual Tracking

Another contribution is that we use the integral image to compute the features for the particle filters based object tracking [131]. With the integral image, we can generate many features at the cost of a few CPU instructions per feature. To

further accelerate the feature generation and improve the posterior sampling, we adopt a coarse-to-fine strategy and quasi-random sampling. In future work, we will investigate how to generate more distinctive features efficiently and how to combine the multiple cues effectively.

Chapter 2

Improved Fast Gauss Transform

The fast multipole method has been called one of the ten most significant numerical algorithms discovered in the 20th century (along with algorithms such as the fast Fourier transform) [30, 14], and won its inventors, V. Rokhlin and L. Greengard, the 2001 Steele prize, in addition to getting Greengard the ACM best dissertation award [54]. The algorithm allows the product of particular dense matrices with a vector to be evaluated approximately (to a specified precision) in $O(N \log N)$ operations, when direct multiplication requires $O(N^2)$ operations. For extremely large problems, the gain in efficiency and memory can be very significant, and enables the use of more powerful modeling approaches that may have been discarded as computationally infeasible in the past.

The FMM represents a fundamental change in the way of designing numerical algorithms, in that it solves the problem approximately, and trades complexity for exactness. However, practically this distinction is usually not important, as in general we need the solution to any scientific problem only to a specified accuracy, and in any case the accuracy specified to the FMM can be arbitrary (e.g., machine precision). Compared to the Fast Fourier Transform (FFT), the FMM does not require that the data be uniformly sampled. It does require the matrix elements to be generated from particular functions, including radial basis functions of various

types which arise naturally in very many applications.

The fast Gauss transform introduced by Greengard and Strain [57, 112] is an important variant of the more general fast multipole method [55, 61]. While the fast multipole method has been successfully in many mathematics and physics domains, the fast Gauss transform is widely applied in many applications of computer vision and pattern recognition [38, 39, 41, 40, 133, 132, 134, 34, 58, 59, 60].

2.1 Fast Multipole Method

To make the thesis self contained, we briefly describe the FMM here. Consider the sum

$$v(y_j) = \sum_{i=1}^{N} u_i \phi_i(y_j), \quad j = 1, \ldots, M. \tag{2.1}$$

Direct evaluation requires $O(MN)$ operations. In the FMM, we assume that the functions ϕ_i can be expanded in multipole (singular) series and local (regular) series that are centered at locations x_* and y_* as follows:

$$\phi(y) = \sum_{n=0}^{p-1} b_n(x_*) S_n\left(y - x_*\right) + \epsilon(p), \tag{2.2}$$

$$\phi(y) = \sum_{n=0}^{p-1} a_n(y_*) R_n\left(y - y_*\right) + \epsilon(p), \tag{2.3}$$

where S_n and R_n respectively are multipole (singular) and local (regular) basis functions, x_* and y_* are expansion centers, a_n, b_n are the expansion coefficients, and ϵ is the error introduced by truncating a possibly infinite series after p terms. The operation reduction trick of the FMM relies on expressing the sum (2.1) using the

series expansions (2.2) and (2.3). Then the reexpansion for (2.3) is:

$$v(y_j) = \sum_{i=1}^{N} u_i \phi_i(y_j) = \sum_{i=1}^{N} u_i \sum_{n=0}^{p-1} c_{ni} R_n \left(y_j - x_*\right), \tag{2.4}$$

for $j = 1, \ldots, M$. A similar expression can be obtained for (2.2). Consolidating the N series into one p term series, by rearranging the order of summations, we get

$$v(y_j) = \sum_{n=0}^{p-1} \left[\sum_{i=1}^{N} u_i c_{ni} \right] R_n(y_j - x_*) = \sum_{n=0}^{p-1} C_n R_n(y_j - x_*)\,. \tag{2.5}$$

The single consolidated p term series (2.5) can be evaluated at all the M evaluation points. The total number of operations required is then $O(Mp + Np) \simeq O(Np)$ for $N \sim M$. The truncation number p depends on the desired accuracy alone, and is independent of M, N.

The functions ϕ_i in the FMM are not valid over the whole domain. So the singular expansions (2.2) are generated around clusters of sources. In a fine-to-coarse pass, the generated coefficients are translated into coarser level singular expansions through a tree data structure by "translation" operators. In a coarse-to-fine pass, the coefficients of the singular expansions at coarser level are converted via a sequence of translations to coefficients of regular expansions at finer levels, then evaluated at each evaluation point.

2.2 Fast Gauss Transform

The fast Gauss transform was introduced in [57] for efficient computation of the weighted sum of Gaussians

$$G(y_j) = \sum_{i=1}^{N} q_i \; e^{-\|y_j - x_i\|^2 / h^2} \tag{2.6}$$

where q_i are the weight coefficients, $\{x_i\}_{i=1,\ldots,N}$ are the centers of the Gaussians (called "sources"), and h is the bandwidth parameter of the Gaussians. The sum of the Gaussians is evaluated at each of the "target" points $\{y_j\}_{j=1,\ldots,M}$. Direct evaluation of the sum at M target points due to N sources requires $O(MN)$ operations.

The original FGT directly applies the FMM idea by using the following expansions for the Gaussian:

$$e^{-\|y-x_i\|^2/h^2} = \sum_{n=0}^{p-1} \frac{1}{n!} \left(\frac{x_i - x_*}{h} \right)^n h_n\left(\frac{y - x_*}{h} \right) + \epsilon(p), \tag{2.7}$$

$$e^{-\|y-x_i\|^2/h^2} = \sum_{n=0}^{p-1} \frac{1}{n!} h_n\left(\frac{x_i - y_*}{h} \right) \left(\frac{y - y_*}{h} \right)^n + \epsilon(p), \tag{2.8}$$

where the Hermite functions $h_n(x)$ are defined by

$$h_n(x) = (-1)^n \frac{d^n}{dx^n} \left(e^{-x^2} \right).$$

The two expansions (2.7) and (2.8) are identical, except that the arguments of the Hermite functions and the monomials (Taylor series) are flipped. The first is used as the counterpart of the multipole expansion, while the second is used as the local expansion. The FGT then uses these expansions and applies the FMM mechanism to achieve its speedup. Conversion of a Hermite series into a Taylor series is achieved via a translation operation. The error bound estimate given by Greengard and Strain [57] is incorrect, and a new and more complicated error bound estimate was presented in [7].

The extension to higher dimensions was done by treating the multivariate Gaussian as a product of univariate Gaussians, applying the series factorizations (2.7) and (2.8) to each dimension. For convenience's sake, we adopt the multi-index

notation of the original FGT paper [57]. A multi-index $\alpha = (\alpha_1, \ldots, \alpha_d)$ is a d-tuple of nonnegative integers. For any multi-index $\alpha \in \mathbf{N}^d$ and any $x \in \mathbf{R}^d$, we have the monomial

$$x^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_d^{\alpha_d}.$$

The length and the factorial of α are defined as

$$|\alpha| = \alpha_1 + \alpha_2 + \ldots + \alpha_d, \qquad \alpha! = \alpha_1! \alpha_2! \cdots \alpha_d!.$$

The multidimensional Hermite functions are defined by

$$h_\alpha(x) = h_{\alpha_1}(x_1) h_{\alpha_2}(x_2) \cdots h_{\alpha_d}(x_d).$$

The sum (2.6) is then equal to the Hermite expansion about center x_*:

$$G(y_j) = \sum_{\alpha \geq 0} C_\alpha h_\alpha \left(\frac{y_j - x_*}{h} \right), \tag{2.9}$$

where the coefficients C_α are given by

$$C_\alpha = \frac{1}{\alpha!} \sum_{i=1}^{N} q_i \left(\frac{x_i - x_*}{h} \right)^\alpha. \tag{2.10}$$

The FGT in higher dimensions is then just an accumulation of the product of the Hermite expansions along each dimension. If we truncate each of the Hermite series after p terms (or equivalently order $p-1$), then each of the coefficients C_α is a d-dimensional matrix with p^d terms. The total computational complexity for a single Hermite expansion is $O((M+N)p^d)$. The factor $O(p^d)$ **grows exponentially** as the dimensionality d increases. Despite this defect in higher dimensions, the FGT is quite effective for two and three-dimensional problems, and has already achieved success in some physics, computer vision and pattern recognition problems [56, 36].

Another serious defect of the original FGT is the use of the box data structure. The original FGT subdivides the space into boxes using a uniform mesh. However, such a simple space subdivision scheme is not appropriate in higher dimensions, especially in applications where the data might be clustered on low dimensional manifolds. First of all, it may generate too many boxes (largely empty) in higher dimensions to store and manipulate. Suppose the unit box in 10 dimensional space is divided into tenths along each dimension, there are 10^{10} boxes which may cause trouble in storage and waste time on processing empty boxes. Secondly, and more importantly, having so many boxes makes it more difficult for searching for nonempty neighbor boxes. Finally, and most importantly the worst property of this scheme is that the ratio of volume of the hypercube to that of the inscribed sphere grows exponentially with dimension. In other words, the points have a high probability of falling into the area inside the box and outside the sphere. The truncation error of the above Hermite expansions (2.7) and (2.8) are much larger near the boundary than near the expansion center, which will bring large truncation errors on most of the points.

In brief, the original FGT suffers from the following two defects that are the motivation behind this chapter:

1. Complexity grows endodontically with dimensionality.

2. The use of the box data structure in the FGT is inefficient in higher dimensions.

2.3 Improved Fast Gauss Transform

2.3.1 A Different Factorization

The defects listed above can be thought as a result of applying the FMM methodology to the FGT blindly. As shown in section 2.1, the FMM was developed for singular potential functions whose forces are long-ranged and nonsmooth (at least locally), hence it is necessary to make use of the tree data structures, multipole expansions, local expansions and translation operators. In contrast, the Gaussian is far from singular — it is infinitely differentiable! There is no need to perform the multipole expansions which account for the far-field contributions. Instead we present a simple new factorization and space subdivision scheme for the FGT. The new approach is based on the fact that the Gaussian, especially in higher dimensions, decays so rapidly that the contributions outside of a certain radius can be safely ignored.

Assuming we have N sources $\{x_i\}$ centered at x_* and M target points $\{y_j\}$, we can rewrite the exponential term as

$$e^{-\|y_j - x_i\|^2/h^2} = e^{-\|\Delta y_j\|^2/h^2}\ e^{-\|\Delta x_i\|^2/h^2}\ e^{2\Delta y_j \cdot \Delta x_i/h^2}, \tag{2.11}$$

where $\Delta y_j = y_j - x_*, \Delta x_i = x_i - x_*$. In expression (2.11) the first two exponential terms can be evaluated individually at either the source points or the target points. The only problem left is to evaluate the last term where sources and targets are entangled. One way of breaking the entanglement is to expand it into the series

$$e^{2\Delta y_j \cdot \Delta x_i/h^2} = \sum_{n=0}^{\infty} \Phi_n(\Delta y_j)\Psi_n(\Delta x_i), \tag{2.12}$$

where Φ_n and Ψ_n are the expansion functions and will be defined in the next section. Denoting $\phi(\Delta y_j) = e^{-\|\Delta y_j\|^2/h^2}$, $\psi(\Delta x_i) = e^{-\|\Delta x_i\|^2/h^2}$, we can rewrite the sum (2.6) as

$$G(y_j) = \sum_{i=1}^{N} q_j \phi(\Delta y_j)\psi(\Delta x_i) \sum_{n=0}^{\infty} \Phi_n(\Delta y_j)\Psi_n(\Delta x_i). \tag{2.13}$$

If the infinite series (2.12) absolutely converges, we can truncate it after p terms so as to obtain a desired precision. Exchanging the summations in (2.13), we obtain

$$G(y_j) = \phi(\Delta y_j) \sum_{n=0}^{p-1} C_n \Phi_n(\Delta y_j) + \epsilon(p), \tag{2.14}$$

$$C_n = \sum_{i=1}^{N} q_i \psi(\Delta x_i)\Psi_n(\Delta x_i). \tag{2.15}$$

The factorization (2.14) is the basis of our algorithm. In the following sections, we will discuss how to implement it in an efficient way.

2.3.2 Multivariate Taylor Expansions

The key issue to speed up the FGT is to reduce the factor p^d in the computational complexity. The factor p^d arises from the way that the multivariate Gaussian is treated as the product of univariate Gaussian functions and expanded along each dimension. To reduce this factor, we treat the dot product in (2.12) as a *scalar variable* and expand it via the Taylor expansion. The expansion functions Φ and Ψ are expressed as multivariate polynomials.

We denote by Π_n^d the space of all real polynomials in d variables of total degree less than or equal to n; its dimensionality is $r_{nd} = \binom{n+d}{d}$. To store, manipulate and

evaluate the multivariate polynomials, we consider the monomial representation of polynomials. A polynomial $p \in \Pi_n^d$ can be written as

$$p(x) = \sum_{|\alpha| \leq n} C_\alpha x^\alpha, \qquad C_\alpha \in \mathbf{R}. \tag{2.16}$$

It is computationally convenient and efficient to stack all the coefficients into a vector. To store all the r_{nd} coefficients C_α in a vector of length r_{nd}, we sort the coefficient terms according to *Graded lexicographic order*. "Graded" refers to the fact that the total degree $|\alpha|$ is the main criterion. Graded lexicographic ordering means that the multi-indices are arranged as

$$(0,0,\ldots,0),(1,0,\ldots,0),(0,1,\ldots,0),\ldots,(0,0,\ldots,1),$$

$$(2,0,\ldots,0),(1,1,\ldots,0),\ldots,(0,0,\ldots,2),\ldots,(0,0,\ldots,n).$$

The power of the dot product of two vectors x and y can be expanded into multivariate polynomial:

$$(x \cdot y)^n = \sum_{|\alpha|=n} \binom{n}{\alpha} x^\alpha y^\alpha, \tag{2.17}$$

where $\binom{n}{\alpha} = \frac{n!}{\alpha_1! \cdots \alpha_d!}$ are the multinomial coefficients. So we have the following multivariate Taylor expansion of the Gaussian functions:

$$e^{2x \cdot y} = \sum_{\alpha \geq 0} \frac{2^{|\alpha|}}{\alpha!} x^\alpha y^\alpha. \tag{2.18}$$

From Eqs.(2.11), (2.14) and (2.18), the weighted sum of Gaussians (2.6) can be expressed as a multivariate Taylor expansions about center x_*:

$$G(y_j) = \sum_{\alpha \geq 0} C_\alpha e^{-\|y_j - x_*\|^2/h^2} \left(\frac{y_j - x_*}{h} \right)^\alpha, \tag{2.19}$$

where the coefficients C_α are given by

$$C_\alpha = \frac{2^{|\alpha|}}{\alpha!} \sum_{i=1}^{N} q_i e^{-\|x_i - x_*\|^2/h^2} \left(\frac{x_i - x_*}{h} \right)^\alpha. \tag{2.20}$$

If we truncate the series after total degree $p-1$, the number of the terms $r_{p-1,d} = \binom{p+d-1}{d}$ is much less than p^d in higher dimensions (as shown in Table 2.1). For instance, when $d = 12$ and $p = 10$, the original FGT needs 10^{12} terms, the multivariate Taylor expansion needs only 293930. For $d \longrightarrow \infty$ and moderate p, the number of terms becomes $O(d^p)$, a substantial reduction.

One of the benefits of the graded lexicographic order is that the expansion of multivariate polynomials can be computed efficiently. For a d-variate polynomial of order n, we can store all terms in a vector of length r_{nd}. Starting from the order zero term (constant 1), we take the following steps recursively. Assume we have already evaluated terms of order $k-1$. Then terms of order k can be obtained by multiplying each of the d variables with all the terms between the variable's *leading term* and the end, as shown in the Figure 2.1. The required storage is r_{nd} and the computations of the terms require $r_{nd} - 1$ multiplications.

2.3.3 Spatial Data Structures

As discussed above, we need to subdivide space into cells and collect the influence of the sources within each cell. The influence on each target can be summarized from its neighboring cells that lie within a certain radius from the target. To efficiently subdivide the space, we need to devise a scheme that adaptively subdivides the space according to the distribution of points. It is also desirable to generate

Table 2.1: Number of terms in d-variate Taylor expansion truncated after order $p-1$.

$p\backslash d$	1	2	3	4	5	6	7	8	9	10	11	12
4	4	10	20	35	56	84	120	165	220	286	364	455
5	5	15	35	70	126	210	330	495	715	1001	1365	1820
6	6	21	56	126	252	462	792	1287	2002	3003	4368	6188
7	7	28	84	210	462	924	1716	3003	5005	8008	12376	18564
8	8	36	120	330	792	1716	3432	6435	11440	19448	31824	50388
9	9	45	165	495	1287	3003	6435	12870	24310	43758	75582	125970
10	10	55	220	715	2002	5005	11440	24310	48620	92378	167960	293930

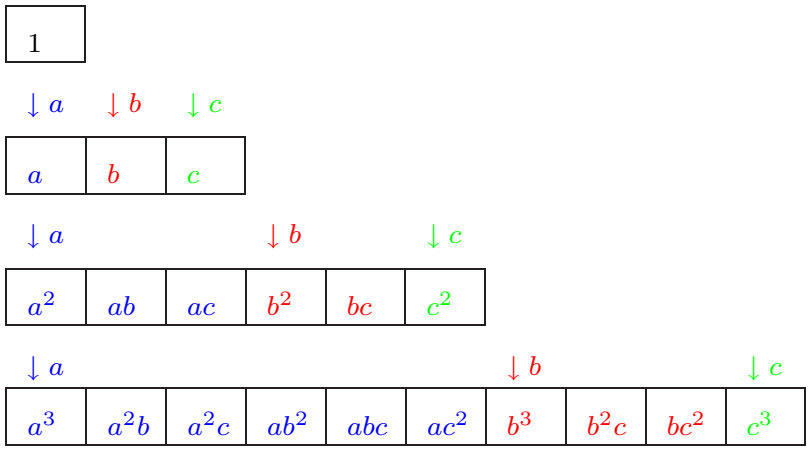


Figure 2.1: Efficient expansion of the multivariate polynomials. The arrows point to the leading terms.

cells as compact as possible.

Based on the above considerations, we model the space subdivision task as a k-center problem, which is defined as follows: given a set of n points and a predefined number of the clusters k, find a partition of the points into clusters $S_1, \ldots, S_k$, and also the cluster centers $c_1, \ldots, c_k$, so as to minimize the cost function — the maximum radius of clusters:

$$\max_i \max_{v \in S_i} \|v - c_i\|.$$

The k-center problem is known to be NP-hard [10]. Gonzalez [53] proposed a very simple greedy algorithm, called *farthest-point clustering*. Initially pick an arbitrary point v_0 as the center of the first cluster and add it to the center set C. Then for $i = 1$ to k do the follows: in iteration i, for every point, compute its distance to the set C: $d_i(v, C) = \min_{c \in C} \|v - c\|$. Let v_i be a point that is farthest away from C, i.e., a point for which $d_i(v_i, C) = \max_v d_i(v, C)$. Add v_i to set C. Report the points $v_0, v_1, \ldots, v_{k-1}$ as the cluster centers. Each point is assigned to its nearest center.

Gonzalez [53] proved that farthest-point clustering is a 2-approximation algorithm which computes a partition with maximum radius at most twice the optimum. The proof uses no geometry beyond the triangle inequity, so it hold for any metric space. Hochbaum and Shmoys [64] proved that the factor 2 cannot be improved unless $P = NP$. The direct implementation of farthest-point clustering has running time $O(nk)$. Feder and Greene [42] give a two-phase algorithm with optimal running time $O(n \log k)$.

The predefined number of clusters k can be determined as follows: run the

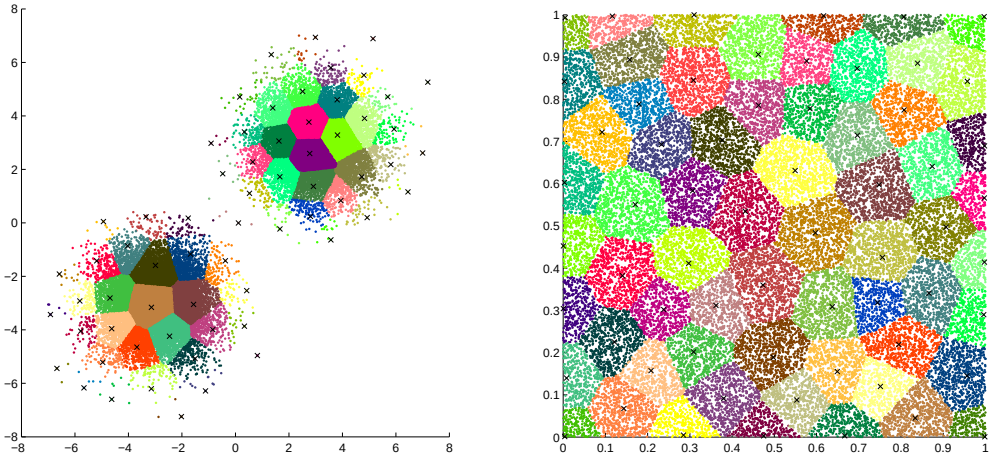


Figure 2.2: The farthest-point algorithm divides 40000 points into 64 clusters (with the centers indicated by the crosses) in 0.48 seconds on a 900MHZ PIII PC. Left: 2 normal distributions; Right: Uniform distribution.

farthest-point algorithm until the maximum radius of clusters decreases to a given distance. In practice, the initial point has little influence on the final radius of the approximation, if the number of points n is sufficiently large. Figure 2.2 displays the results of farthest-point algorithm. In two dimensions, the algorithm leads to a Voronoi tessellation of the space. In three dimensions, the partition boundary resembles the surface of a crystal.

2.3.4 The Algorithm

The improved fast Gauss transform consists of the following steps:

Step 1. Assign the N sources into K clusters using the farthest-point clustering algorithm such that the radius is less than $h\rho_x$.

Step 2. Choose p sufficiently large such that the error estimate (2.23) is less than the

desired precision ϵ.

Step 3. For each cluster S_k with center c_k, compute the coefficients given by the expression (2.20):

$$C_\alpha^k = \frac{2^{|\alpha|}}{\alpha!} \sum_{x_i \in S_k} q_i e^{-\|x_i - c_k\|^2/h^2} \left(\frac{x_i - c_k}{h} \right)^\alpha .$$

Step 4. Repeat for each target y_j, find its neighbor clusters whose centers lie within the range $h\rho_y$. Then the sum of Gaussians (2.6) can be evaluated by the expression (2.19):

$$G(y_j) = \sum_{\|y_j - c_k\| \le h\rho_y} \sum_{|\alpha|<p} C_\alpha^k e^{-\|y_j - c_k\|^2/h^2} \left(\frac{y_j - c_k}{h} \right)^\alpha .$$

2.3.5 Complexity and Error Bounds

The amount of work required in step 1 is $O(NK)$ (for large K, we can use Feder and Greene's $O(N \log K)$ algorithm [42] instead). The amount of work required in step 3 is of $O(N\ r_{pd})$. The work required in step 4 is $O(Mn\ r_{pd})$, where n is the maximum number of the neighbor clusters for each target. For most nonparametric statistics, computer vision and pattern recognition applications, the precision required is moderate, we can get small K and small r_{pd}. Since $n \le K$, the improved fast Gauss transform achieves linear running time. The algorithm needs to store the K coefficients of size r_{pd}, so the storage complexity is reduced to $O(Kr_{pd})$.

The error in the above algorithm arises from two sources. The first is due to truncation of the Taylor series in step 3, and the other due to cutoff of the far field

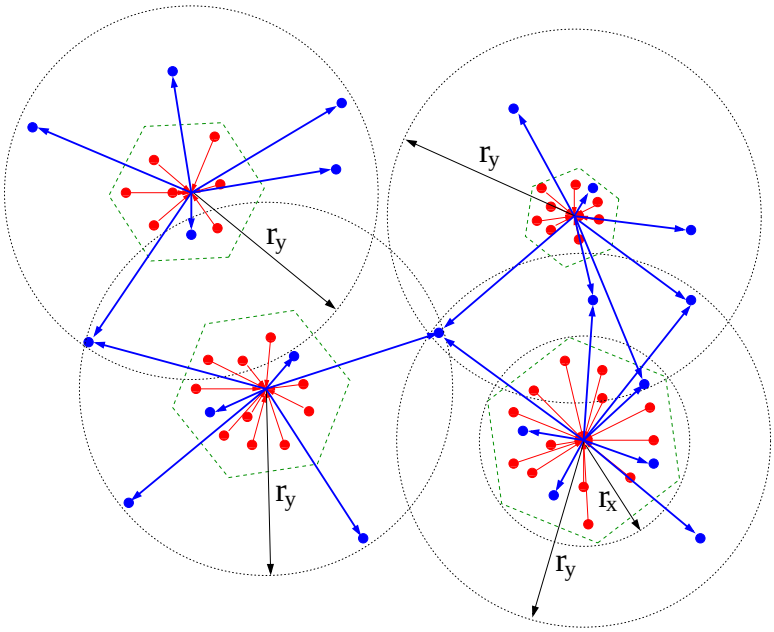


Figure 2.3: The improved fast Gauss transform. The sources (red dots) are grouped into k clusters by the farthest-point clustering algorithm. r_x is the radius of the farthest-point clustering algorithm. The contributions on the target (blue dot) outside of the cutoff radius r_y are ignored.

contributions in step 4. The error $E_T(x,p)$ due to truncating the series at source point x after order $p-1$ satisfies the bound:

$$\begin{aligned}
|E_T(x,p)| &\le e^{-(\|\Delta x\|^2+\|\Delta y\|^2)/h^2}\ \frac{1}{p!}\ e^{2\Delta x\cdot\Delta y/h^2}\ \left(\frac{2\Delta x\cdot\Delta y}{h^2}\right)^p \\
&\le e^{-(\|\Delta x\|^2+\|\Delta y\|^2)/h^2}\ \frac{2^p}{p!}\ e^{2\|\Delta x\|\|\Delta y\|/h^2}\ \left(\frac{\|\Delta x\|\|\Delta y\|}{h^2}\right)^p \qquad (2.21)\\
&\le \frac{2^p}{p!}\ \left(\frac{r_x r_y}{h^2}\right)^p = \frac{2^p}{p!}\ \rho_x^p\rho_y^p.
\end{aligned}$$

where $\Delta x_i = x_i - x_*$ and $\Delta y_j = y_j - x_*$, r_x is the upper bound of $\|\Delta x\|$, and r_y is the upper bound of $\|\Delta y\|$. We also denote the ratios $\rho_x = r_x/h$ and $\rho_y = r_y/h$. The Cauchy inequality

$$\Delta x\cdot\Delta y \le \|\Delta x\|\|\Delta y\|,$$

and the inequality

$$2\|\Delta x\|\|\Delta y\| \le \|\Delta x\|^2 + \|\Delta y\|^2,$$

were used in the above error bound analysis.

The cutoff error $E_C(r_y)$ due to ignoring contributions outside of radius r_y from target point y satisfies the bound:

$$|E_C(r_y)| \le e^{-r_y^2/h^2} = e^{-\rho_y^2}. \qquad (2.22)$$

The total error at any target point y satisfies the bound:

$$|E(y)| \le \left|\sum q_j E_T(x,p)\right| + \left|\sum q_j E_C(r_y)\right| \le Q\left(\frac{2^p}{p!}\ \rho_x^p\rho_y^p + e^{-\rho_y^2}\right). \qquad (2.23)$$

where $Q = \sum |q_j|$.

From above error bound, we can see that p, ρ_x and ρ_y together control the convergence rate. The larger p and ρ_y, the smaller ρ_x, the algorithm converges

faster. But the cost of computation and storage increases at the same time. There is always a tradeoff between the speed and the precision. The above error bound is much simpler than the error estimate in [7]. Another interesting fact about the error bound is that it is independent of the dimensionality.

For bandwidth comparable to the range of the data, we can increase the steps of the farthest-point algorithm to decrease the radius r_x. The radius of farthest-point algorithm always decreases as the algorithm progresses. By this way, we can make $\rho_x \rho_y < 1$, so the error bound (2.23) always converges.

For very small bandwidth (for instance $\rho_x > 10$), the interaction between the sources and targets are highly localized. We can set $p = 0$ which means we directly accumulate the contributions of the neighboring sources and there is no series expansion. All we need is a good nearest neighbor search algorithm [87, 88].

2.4 Numerical experiments

The first experiment compares the performance of our proposed algorithm with the original fast Gauss transform. Since there is no practical fast Gauss transform in higher dimensions available, we only make comparisons in three dimensions. The sources and targets are uniformly distributed in a unit cube. The weights of the sources are uniformly distributed between 0 and 1. The bandwidth of the Gaussian is $h = 0.2$. We set the relative error bound to 2 which is reasonable for most kernel density estimation in nonparametric statistics where Gauss transform plays an important role, because the estimated density function itself is an approxima-

tion. Table 2.2 reports the CPU times using direct evaluation, the original fast Gauss transform (FGT) and the improved fast Gauss transform (IFGT). All the algorithms are programmed in C++ and were run on a 900MHz PIII PC. We find that the running time of the IFGT grows linearly as the number of sources and targets increases, while the direct evaluation and the original FGT grow quadratically, though the original FGT is faster than the direct evaluation. The poor performance of the FGT in 3D is also reported in [36]. This is probably due to the fact that the number of boxes increases significantly by a uniform space subdivision in 3D. The cost to compute the interactions between the boxes grows quadratically. The farthest-point algorithm in the IFGT generates a much better space subdivision and reduces the number of boxes greatly. The multivariate Taylor expansion also reduces the computational cost by a factor 4.3 in 3D (the factor is for order 7, and larger factors in higher dimensions).

The second experiment is to examine the performance of IFGT in higher dimensions. We randomly generate the source and target points in a unit hypercube according to a uniform distribution. The weights of the sources are uniformly distributed between 0 and 1. The bandwidth is set to $h = 1$. The results are shown in Fig. 2.4. We compared the running time of the direct evaluation to the IFGT with $h = 1$ and $N = M = 100, \ldots, 10000$. The comparisons are performed in dimensions from 4 to 10 and results in dimensions 4, 6, 8, 10 are reported in Figure 2.4. From the figure we notice that the running time of the direct evaluation grows quadratically with the size of points. The running time of the IFGT grows linearly with the size of the points. In 4, 6, 8, 10 dimensions, the IFGT takes 56ms, 406ms, 619

Table 2.2: Running times in milliseconds for direct evaluation, fast Gauss transform and improved fast Gauss transform in three dimensions.

Case	$N = M$	Direct	FGT	IFGT
1	100	2.9	5.5	4.6
2	200	11.4	13.0	12.5
3	400	46.1	37.0	21.1
4	800	184.2	121.8	33.2
5	1600	740.3	446.0	68.1
6	3200	2976.2	1693.8	132.8
7	6400	17421.4	6704.3	263.0
8	12800	68970.2	26138.6	580.2
9	25600	271517.9	103880.8	1422.0

ms, 1568ms to evaluate the sums on 10000 points, while it takes 35 seconds for a direct evaluation. The maximum relative absolute error as defined in [57] increases with the dimensionality but not with the number of points. The worst relative error occurs in dimension 10, and is below 10^{-3}. We can see that for a 10D problem involving more than 700 Gaussians, the IFGT is faster than direct evaluation, while for a 4D problem the IFGT is faster from almost the outset. We also tested our algorithm on the normal distributions with mean zero, variance one of sources and targets. All data were scaled into unit hypercube. The results are shown in Figure 2.4. We find that the running time is similar to the case of uniform distribution, while the error is much less than the case of uniform distribution.

The third experiment is to examine the error bounds of the IFGT. 1000 source

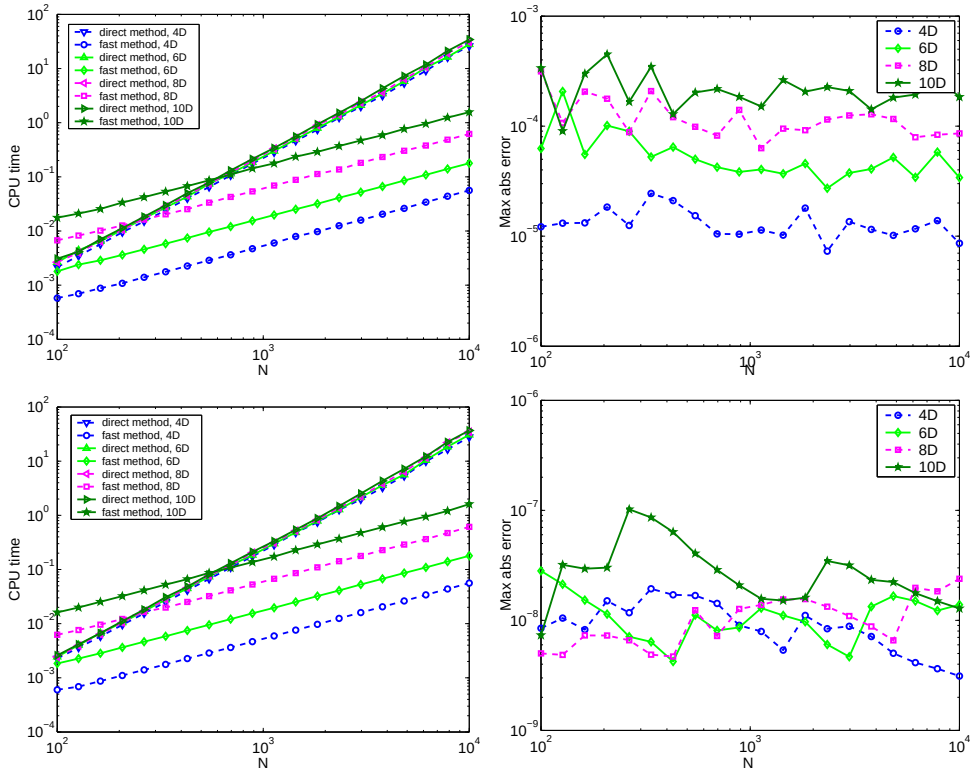


Figure 2.4: The running times in seconds (Left column) and maximum absolute errors (Right column) of the IFGT ($h = 1$) v.s. direct evaluation in dimensions $4, 6, 8, 10$ on uniform distribution (Top row) and normal distribution (Bottom row).

points and 1000 target points in a unit hypercube are randomly generated from a uniform distribution. The weights of the sources are uniformly distributed between 0 and 1. The bandwidth is set to $h = 0.5$. We fix the order of the Taylor series $p = 10$, the radius of the farthest-point clustering algorithm $r_x = 0.5h$, and the cutoff radius $r_y = 6h$, then we vary p, r_x and r_y, and repeat the experiments in 4 dimensions and 6 dimensions respectively. The comparison between the real maximum absolute errors and the estimated error bounds is shown in Figure 2.5. The estimated error bounds are almost optimal up to a constant factor. The normalized error bounds

w.r.t. the order p and the radius r_x upbound the curves of the real errors, which indicates the constant factor for them is roughly the number of the sources. In the case of small cutoff radius, the constant factor is smaller because influence on each target point is highly localized so the sources points far away seem to vanished. The estimated error bounds are useful for choosing the parameters of the IFGT.

2.5 Conclusions

We have proposed an improved fast Gauss transform that leads to a significant speedup with a major reduction in the amount of memory required in higher dimensions. A multivariate Taylor expansion is applied to the improved fast Gauss transform which substantially reduces the number of the expansion terms in higher dimensions. The k-center algorithm is utilized to efficiently and adaptively subdivide the higher dimensional space according to the distribution of the points. A simpler and more accurate error estimate is reported, due to the simplification made by the new Taylor expansion and space subdivision schemes. The improved fast Gauss transform is capable of computing the Gauss transform in dimensions as high as tens which commonly occur in nonparametric statistics and pattern recognition. The behavior of the algorithm in very high dimensional space (such as up to several hundreds) will be studied and reported.

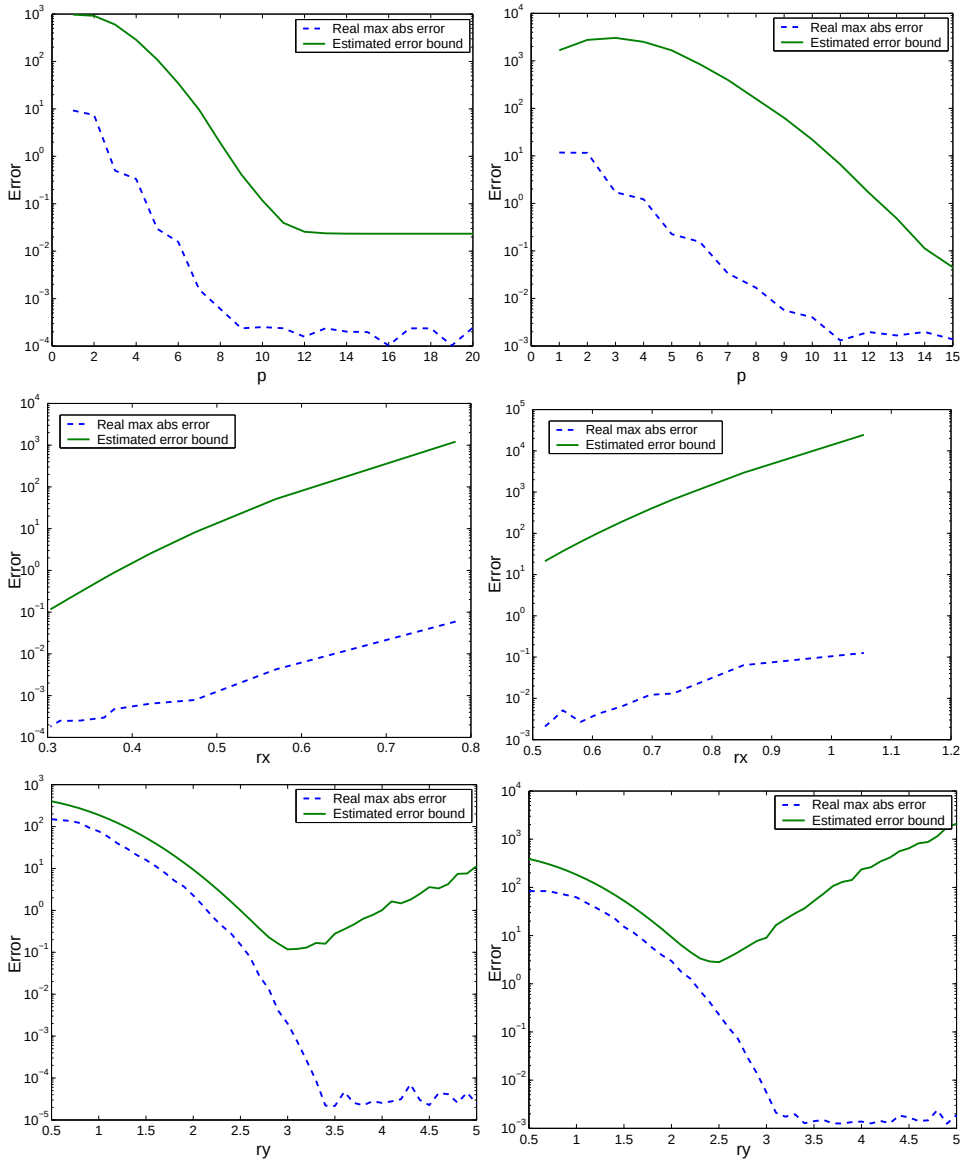


Figure 2.5: The comparison between the real maximum absolute errors and the estimated error bounds w.r.t. the order of the Taylor series p (Top row), the radius of the farthest-point clustering algorithm r_x (Middle row), and the cutoff radius r_y (Bottom row). The uniformly distributed sources and target points are in 4 dimensions (Left column) and in 6 dimensions (Right column).

Chapter 3

Efficient Kernel Density Estimation

The statistical methods in computer vision and pattern recognition require knowledge of the probability density functions. There exist two categories of methods to estimate the probability density functions. The parametric methods make assumptions regarding the form of the density functions, such as the Gaussian distribution or mixture of Gaussians. In most computer vision and pattern recognition applications, however, the form of the underlying density functions is not available.

There are many methods that have been used for estimating the probability density functions. Historically the histogram approach was used and still plays an important role in many application nowadays. But the histogram approach is limited to low dimensional spaces because the number of bins increases exponentially with dimensionality. To overcome this difficulty, the kernel density estimation (KDE), first introduced by Rosenblatt [102], then studied in detail by Parzen [91], can be applied to estimate the probability density functions. In this technique the density function is estimated by a sum of kernel functions (typically Gaussians) centered at the data points. A bandwidth associated with the kernel function is chosen to control the smoothness of the estimated densities. In general, more data points allow a narrower bandwidth and a better density estimate. On the other hand, the number of samples needed may be very large and much greater than would be

required for parametric models. Moreover, the demand for large number of samples grows rapidly with the dimension of the feature space. Given N source data points, the direct evaluation of densities at M target points takes $O(MN)$ time. The large dataset also leads to severe requirements for computational time and/or storage.

Various methods have been proposed to make the process of kernel density estimation more efficient. The nearest neighbor searching and/or branch and bound methods achieve the computational saving using tree-based data structures [97, 29, 48, 74]. The fast Fourier transform (FFT) evaluates density estimates on gridded data which, however, are unavailable for most applications [108]. Recently the fast multipole method (FMM) and fast Gauss transform (FGT) have been used to reduce the computational time of kernel density estimation to linear order, where the data are not necessarily on grids [57, 36].

As faster computers and better video cameras become cheaper, the collection of sufficient data is becoming possible, which results in a steady increase in the size of the dataset and the number of the features. Unfortunately the existing approaches including the fast Gauss transform suffer from the curse of dimensionality. The complexity of computation and storage of the FGT grows exponentially with dimension. In Chapter 2, we presented an improved fast Gauss transform (IFGT) to efficiently evaluate the sum of Gaussians in higher dimensions, which can be used to speed up the kernel density estimation even in higher dimensions.

In this chapter, we show how the IFGT can be applied to the kernel density estimation. Specifically the mean shift algorithm [49, 17, 22] is chosen as a case study for the IFGT. Mean-shift analysis is a relatively new but important clustering

approach originally invented by Fukunaga and Hostetler [49] which they called a "valley-seeking procedure". In spite of its excellent performance, it had been nearly forgotten until Cheng [17] extended it and introduced it to the image analysis community. Recently Comaniciu and Meer [21, 22, 24] successfully applied it to image segmentation and object tracking. The recently established relationship between mean-shift and M-estimator further extends its applications [66, 22]. However, the mean shift algorithm suffers from the quadratic computational complexity, especially in higher dimensions. The IFGT successfully reduced the computational complexity into linear time.

Mean-shift is based on the KDE which estimates the gradient of the density function in feature space. Advantages of feature-space methods are the global representation of the original data and the excellent tolerance to noise [33]. When a density function in feature space has peaks and valleys, it is desirable to divide data points into clusters according to the valleys of the point densities, because such boundaries in feature space are mapped back to much more natural segmentation boundaries.

The mean-shift procedure consists of two steps: the estimation of the gradient of the density function, and the utilization of the results to form clusters. The gradient of the density function is estimated by the KDE. Then starting from each sample point, the mean-shift procedure iteratively finds a path along the gradient direction away from the valleys and towards the nearest peak.

In this chapter, we focus on mean-shift based image segmentation using improved fast Gauss transform. However such methodology can be applied to nor-

malized cuts [106] in the same way. Moreover, the anisotropic diffusion, bilateral filtering [115, 12, 35] can be accelerated under the same framework.

3.1 Mean-shift Based Image Segmentation

Image segmentation using the mean shift analysis is chosen as a case study for the efficient kernel density estimation. Mean shift is a clustering technique based on kernel density estimation, which is very effective and robust for the analysis of complex feature spaces. The mean shift procedure employing a Gaussian kernel converges to the stationary point following a smooth trajectory, which is theoretically important for convergence [22]. In practice, the quality of the results almost always improves when the Gaussian kernel is employed. Despite its superior performance, the Gaussian kernel is not as widely used in mean shift as it should be. In part this may be due to the high computational costs which we try to alleviate in this chapter.

Given n data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in the d-dimensional space R^d, the *kernel density estimator* with kernel function $K(\mathbf{x})$ and a window bandwidth h, is given by [102, 91, 33]

$$\hat{p}_n(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right), \tag{3.1}$$

where the d-variate kernel $K(\mathbf{x})$ is nonnegative and integrates to one. The Gaussian kernel is a common choice.

The mean shift algorithm is a steepest ascent procedure which requires esti-

mation of the density gradient:

$$\begin{aligned}\nabla \hat{p}_{h,K}(\mathbf{x}) &= \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{x}) g\left(\left\| \frac{\mathbf{x} - \mathbf{x}_i}{h} \right\|^2 \right) \\ &= c_{k,g} \hat{p}_{h,G}(\mathbf{x}) \left[\frac{\sum_{i=1}^{n} \mathbf{x}_i g\left(\left\| \frac{\mathbf{x}-\mathbf{x}_i}{h} \right\|^2 \right)}{\sum_{i=1}^{n} g\left(\left\| \frac{\mathbf{x}-\mathbf{x}_i}{h} \right\|^2 \right)} - \mathbf{x} \right],\end{aligned} \tag{3.2}$$

where $g(x) = -k'_N(x) = \frac{1}{2} k_N(x)$ which can in turn be used to define a Gaussian kernel $G(\mathbf{x})$ such that $G(\mathbf{x}) = g(\|\mathbf{x}\|^2)$. The kernel $K(\mathbf{x})$ is called the shadow of $G(\mathbf{x})$ [17]. Both have the same expression. $\hat{p}_{h,G}(\mathbf{x})$ is the density estimation with the kernel G. $c_{k,g}$ is the normalization coefficient. The last term is the *mean shift*

$$\mathbf{m}(\mathbf{x}) = \frac{\sum_{i=1}^{n} \mathbf{x}_i g\left(\left\| \frac{\mathbf{x}-\mathbf{x}_i}{h} \right\|^2 \right)}{\sum_{i=1}^{n} g\left(\left\| \frac{\mathbf{x}-\mathbf{x}_i}{h} \right\|^2 \right)} - \mathbf{x}, \tag{3.3}$$

which is proportional to the normalized density gradient and always points toward the steepest ascent direction of the function. The mean shift algorithm iteratively performs the following two steps till it reaches the stationary point:

- Computation of the mean shift vector $\mathbf{m}(\mathbf{x}^k)$.
- Updating the current position $\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{m}(\mathbf{x}^k)$.

If the Gaussian kernel is applied, the denominator is a uniform weighted sum of Gaussians. The numerator in expression (3.3) is a weighted sum of Gaussians except that the weights are vectors $\mathbf{x}_i$. We can denote the numerator as

$$f_j(\mathbf{x}) = \sum_{i=1}^{n} x_{ij} g\left(\left\| \frac{\mathbf{x} - \mathbf{x}_i}{h} \right\|^2 \right), \tag{3.4}$$

where $\mathbf{x}_i = \{x_{ij}\}, j = 1, \ldots, d$. So both the denominator and the numerator can be evaluated by the improved fast Gauss transform as $d + 1$ independent weighted

sums of Gaussians. The computation has been further reduced because they share the same space subdivisions and series expansions.

3.2 Experimental Results

This experiment is to apply the improved fast Gauss transform to the mean shift algorithm. We first transform the images to $L^*u^*v^*$ color space and normalize to a unit cube. Then we apply the mean shift algorithm with $h = 0.1$ to all the points in the 3D color space. After 5 iterations, the convergence points are grouped by a simple k-means algorithm [33]. We do not perform any postprocessing procedure as in [22]. The code is written in C++ with Matlab interfaces and run on a 900MHz PIII PC. The results are shown in Fig. 3.1. The running time of the mean shift in seconds and the sizes of the images are shown in Table 3.1. The speed of our implementation is at least as fast as any reported. We find that the mean shift algorithm with the improved fast Gauss transform already achieves clear boundaries without any postprocessing. This is partly because that we apply the mean shift algorithm to all feature points without subsampling the feature space as in [22]. This leads to easily distinguishable valleys in our estimated densities. Another reason is that in our method the density evaluation at each target point has contributions from a much larger neighborhood because of the Gaussian kernel, which generates a smoother and better density estimate.

Figure 3.1: Segmentation results: (Right Column) The original images. (Left Column) Segmented images labeled with different colors. (Top Row) *House* image. (Second Row) *Cooking* image. (Third Row) *Base Dive* image. (Bottom Row) *Zebra* image.

Table 3.1: Image sizes *v.s.* the running time of the mean shift.

	House	*Cooking*	*Base Dive*	*Zebra*
Size	255x192	204x153	432x294	481x321
Time (s)	3.343	2.204	7.984	12.359

3.3 Discussion

The kernel density estimation is a very important and effective approach in computer vision and pattern recognition. However the slow evaluation is a major drawback which prevents its application from many time-demanding problems. The acceleration of the KDE is traditionally based on the fast nearest neighbor searching using various data structures such as trees, or hashing tables [88]. Such data structures are usually complex and involved to build up. Moreover, their performances degrade exponentially when the dimensionality increases. The fast Gauss transform alleviates the demand on the spatial data structure by the series expansions. However the number of box data structures and Hermite expansion terms in the original FGT grows exponentially with the dimensions, so it is not suitable in dimensions above three. Our improved FGT fixes these two defects and speeds up the KDE even in higher dimensions.

The mean-shift algorithm employs KDE to estimate the probabilistic density function. To deal with the high amount of computations, the data points are randomly subsampled which results in the loss of resolution. The robust anisotropic diffusion and the bilateral filtering are based on the KDE. In [35], in order to speed up the computation, the feature space is uniformly divided into brackets and data

are linearly interpolated (called piecewise-linear approximation). From the point view of improved FGT, such approximation is a special case with $p = 1$ and uniform space subdivision.

Chapter 4

Real-Time Object Tracking

Object tracking is a common vision task to find and follow moving objects between consecutive frames, which is important for many computer vision applications such as human-computer interaction, surveillance, smart rooms and medical imaging. A variety of tracking algorithms have been proposed and implemented to overcome difficulties that arise from noise, occlusion, clutter, and changes in the foreground objects being tracked or in the background environment. Region-based methods typically align the tracked regions between the successive frames by minimizing a cost function [69, 5, 62]. Feature-based approaches extract features (such as intensity, colors, edges, contours) and use them to establish correspondence between model images and target images [70, 43, 24]. Model-based tracking algorithms incorporate *a priori* information about the tracked objects to develop representations such as projected shape, skin complexion, body blobs, kinematic skeleton and silhouette [135, 127, 15, 109, 18]. Appearance-based approaches apply recognition algorithms to learn the objects either in the eigenspace or in the kernel space. The trained systems are used to search for the targets in image sequences [13, 2, 126].

Many of these approaches employ a statistical description of the region or the pixels to perform the tracking. The tracked objects can be described using either parametric or nonparametric representations. In a parametric framework,

the objects or persons are typically fitted by Gaussian models or via a mixture of Gaussians [127]. A nonlinear estimation problem has to be solved to obtain the number of Gaussians and their parameters. However, the common parametric forms rarely fit the multimodal complex densities in practice, and are problematic when the fitted distributions are multidimensional. In contrast, nonparametric density estimation techniques [91, 33] allow representation of complex densities just by using the data. They have been successfully applied to object tracking [24, 37]. The conceptually simplest density estimation approach is to build a histogram and use it to establish the correspondences between the model image and the target image [43, 24]. The histogram is very flexible and robust for tracking deformable and nonrigid objects. However histogramming is only suitable for low dimensional spaces because as the number of dimensions increase, the number of bins grows exponentially. In contrast, given sufficient samples, kernel density estimation works well both in low and high dimensions, and has successfully been applied to object tracking [37].

To build a matching of the objects across frames, many tracking algorithms use measures of "similarity" or "distance" between the two regions, feature vectors, or distributions. The sum of squared differences (SSD) assumes "constant brightness" from frame to frame [69, 62], which is liable to fail with noise, deformation or occlusion. The Kullback-Leibler divergence, Bhattacharyya distance and other probabilistic distance functions are employed to measure the similarity between frames [24, 37]. All these information-theoretic distance measures require an estimate of the conditional probability density function and its numerical integration. When such measures are used by the mean shift algorithm or other gradient based

methods, the evaluation of their gradient functions is often involved, numerically unstable and computationally expensive, especially in high dimensions.

The mean shift algorithm, originally invented by Fukunaga and Hostetler [49], was successfully applied to computer vision applications by Comaniciu [22, 24]. It is an effective gradient-based optimization technique for finding the target location but has two difficulties. First, the kernel-based densities are expensive to evaluate. Second, the classically used similarity measures between the distributions in the model and target images are unwieldy, and computationally even more expensive to evaluate than the density.

In this chapter we address these difficulties by presenting an object tracking algorithm that uses a simple symmetric similarity function between kernel density estimates of the model and target distributions. In our formulation we use the joint spatial-feature formulation of [37], and consider both feature vectors and pixel locations as probabilistic random variables. The density is estimated in the joint feature-spatial space using radial-basis kernel functions which measure the affinity between points and provide a better outlier rejection property. The joint feature-spatial spaces impose a probabilistic spatial constraint on the tracked region and provide an accurate representation of the tracked objects. The similarity measure we use is symmetric and is the expectation of the density estimates centered on the model (target) image over the target (model) image. The mean shift algorithm is used to track objects by iteratively maximizing this similarity function. To alleviate the quadratic complexity of the density estimation, we employ Gaussian kernels and the improved fast Gauss transform (FGT) [133] to reduce the computations to linear

order.

4.1 Image Representation

The distribution of features and pixels of the tracked objects are represented as probability distribution functions over joint feature-spatial spaces. Pixels in the spatial domain are mapped into points in a multidimensional feature space. Such a mapping is used to characterize the tracked objects and is usually nonlinear. A good feature space will greatly relieve difficulties in distinguishing objects from the background and provide tolerance to the noise [107]. The most commonly used features are image intensity, colors, edges, texture, wavelet filter response, etc. The associated spatial space enhances the feature space by imposing the constraint of spatially continuity in a statistical way.

Suppose we are given two images, with one designated as the "model image" that includes the tracked objects, while the other is the "target image" in which we need to find the objects. The sample points in the model image are denoted by $I_x = \{\mathbf{x}_i, \mathbf{u}_i\}_{i=1}^{N}$, where $\mathbf{x}_i$ is the 2D coordinates and $\mathbf{u}_i$ is the corresponding feature vector. The sample points in the target image are $I_y = \{\mathbf{y}_j, \mathbf{v}_j\}_{j=1}^{M}$, encoding the the 2D coordinates and the corresponding feature vector.

The structure of the joint feature-spatial spaces is generally complex and can be analyzed only by nonparametric methods. The probability density function of the joint feature-spatial spaces can be estimated from the sample points by the kernel density estimation [91, 33]. In pattern recognition and computer vision, the

following radial-basis function (RBF) kernel (symmetric, positive-definite) is widely used [91, 95, 104, 96]:

$$k(\mathbf{x}, \mathbf{x}') = k(\left\| \frac{\mathbf{x} - \mathbf{x}'}{h} \right\|^2), \tag{4.1}$$

where $k(x)$ is the *profile* of the kernel, and h is the *bandwidth*. The important RBF kernel — Gaussian kernel in d dimensions is

$$k(\mathbf{x}, \mathbf{x}') = \frac{1}{(2\pi)^{d/2} h^d} e^{-\|\mathbf{x}-\mathbf{x}'\|^2/2h^2}, \tag{4.2}$$

which is supported by many results from psychology and learning theory [95, 96].

Given the sample points and the RBF kernel function $k(x)$, the probability density function of the model image can be estimated in the feature space as

$$\hat{p}_x(\mathbf{u}) = \frac{c_d}{N} \sum_{i=1}^{N} k(\left\| \frac{\mathbf{u} - \mathbf{u}_i}{h} \right\|^2). \tag{4.3}$$

where c_d is the normalization constant factor. In our framework, this normalization factor can be neglected, because we use the mean-shift algorithm where the same kernel functions are used in both numerator and denominator. For the sake of simplicity, we drop this factor in the rest of this discussion.

Usually the exterior points of a region are less reliable than the interior points. To combat noise and improve robustness, we regularize the probability density function (4.3) by smoothing it with another RBF kernel $w(x)$ in the spatial domain [24]. Then the spatially-smoothed probability density function of the model image centered at $(\mathbf{x}, \mathbf{u})$ can be estimated in the joint feature-spatial space as

$$\hat{p}_x(\mathbf{x}, \mathbf{u}) = \frac{1}{N} \sum_{i=1}^{N} w(\left\| \frac{\mathbf{x} - \mathbf{x}_i}{\sigma} \right\|^2) k(\left\| \frac{\mathbf{u} - \mathbf{u}_i}{h} \right\|^2). \tag{4.4}$$

Similarly the spatially-smoothed probability density function of the target image centered at $(\mathbf{y}, \mathbf{v})$ can be estimated as

$$\hat{p}_y(\mathbf{y}, \mathbf{v}) = \frac{1}{M} \sum_{j=1}^{M} w(\left\| \frac{\mathbf{y} - \mathbf{y}_j}{\sigma} \right\|^2) k(\left\| \frac{\mathbf{v} - \mathbf{v}_j}{h} \right\|^2), \tag{4.5}$$

where σ and h are the bandwidths in the spatial and feature spaces. We also absorb the normalization constants into the kernels for convenience.

4.2 Similarity Between Distributions

Once we have the probability density functions of two distributions, we need a similarity (or dissimilarity) function to measure the affinity between groups of points or distributions. There are many similarity measures between distributions proposed in statistics and pattern recognition [28, 122, 98]. A conceptually simple similarity measure is the sum of squared differences (SSD) [69, 62]. Several probabilistic distance measures have been proposed [28, 122] and some have been applied to tracking. In [24], the Bhattacharyya coefficient is employed as the similarity measure. The Kullback-Leibler divergence is used as similarity measure in [119, 37]. All of these information-theoretic distance measures require an estimate of the probability density function and its numerical integration. Their gradient functions are often involved and numerically unstable, especially in high dimensions.

In this chapter, we define the similarity between two distributions as the expectation of the spatially-smoothed density estimates over the model or target image. Suppose we have two distributions with samples $I_x = \{\mathbf{x}_i, \mathbf{u}_i\}_{i=1}^{N}$ and $I_y = \{\mathbf{y}_j, \mathbf{v}_j\}_{j=1}^{M}$, where $\mathbf{x}_i$ and $\mathbf{y}_j$ are 2D coordinates, $\mathbf{u}_i$ and $\mathbf{v}_j$ are feature vectors,

the center of sample points in the model image is $\mathbf{x}_*$, and the current center of the target points is $\mathbf{y}$, the spatially-smoothed similarity between I_x and I_y is

$$J(I_x, I_y) = \frac{1}{M} \sum_{j=1}^{M} \hat{p}_x(\mathbf{y}_j, \mathbf{v_j}), \tag{4.6}$$

or symmetrically

$$J(I_y, I_x) = \frac{1}{N} \sum_{i=1}^{N} \hat{p}_y(\mathbf{x}_i, \mathbf{u}_i). \tag{4.7}$$

From equations (4.4) and (4.5), both equations (4.6) and (4.7) can be rewritten as

$$J(I_x, I_y) = \frac{1}{MN} \sum_{i=1}^{N} \sum_{j=1}^{M} w(\left\| \frac{\mathbf{x}_i - \mathbf{y}_j}{\sigma} \right\|^2) k(\left\| \frac{\mathbf{u}_i - \mathbf{v}_j}{h} \right\|^2). \tag{4.8}$$

The similarity function (4.8) can be interpreted as the expectation of the spatially-smoothed density estimates over the model image.

We normalize the data along each dimension and use fixed bandwidth for simplicity. Variable and adaptive bandwidth can be applied to the similarity function (4.8) and will give better performance. The spatial smoothing can also be improved by considering the background information and the shape of the region.

The similarity measure (4.8) is symmetric and bounded by zero and one, but violates the triangle inequality which means the similarity measure is non-metric. Often distance functions that are robust to outliers or to noise disobey the triangle inequality [73].

If we set $\sigma \to 0$, then $w(x)$ becomes a delta function. The similarity function reduces to the robust error function

$$J(I_x, I_y) = \frac{1}{N} \sum_{i=1}^{N} k(\left\| \frac{\mathbf{u}_i - \mathbf{v}_i}{h} \right\|^2). \tag{4.9}$$

Minimizing error function (4.9) results in a robust version of Lucas-Kanade algorithm [85, 3].

If we set $\sigma \to \infty$, then $w(x)$ becomes a uniform function. The joint feature-spatial probability density estimations (4.4) and (4.5) reduce to

$$\hat{p}_x(\mathbf{x}, \mathbf{u}) = \frac{1}{N} \sum_{i=1}^{N} k(\| \frac{\mathbf{u} - \mathbf{u}_i}{h} \|^2). \tag{4.10}$$

and

$$\hat{p}_y(\mathbf{y}, \mathbf{v}) = \frac{1}{M} \sum_{j=1}^{M} k(\| \frac{\mathbf{v} - \mathbf{v}_j}{h} \|^2), \tag{4.11}$$

which are the kernel density estimations in the feature spaces. Minimizing the similarity function (4.8) results in the histogram tracking where the spatial constraints are lost [11, 43].

The similarity measure (4.8) is directly computed from the sample points. The affinities between all pairs of sample points are considered based on their distances and exact correspondence is not necessary, which is more robust than the template matching or sum of squared differences (SSD). Furthermore, the sample points are sparse in the high dimensional feature space. It is difficult to get an accurate density estimation or histogram which will cause the similarity measures such as Kullback-Leibler divergence and Bhattacharyya coefficient to become unstable. The effectiveness of similarity measure (4.8) in high dimensional space is well explained by the theories developed for support vector machines [104, 96].

The similarity function (4.8) is non-metric. However, it can be shown that its negative natural logarithm

$$L(I_x, I_y) = -\log J(I_x, I_y) \tag{4.12}$$

is a probabilistic distance, provided we have sufficient samples, so that the kernel density estimate converges to the true probability density function [28].

To compare our similarity measure with other commonly used probabilistic distances, specifically Bhattacharyya distance and Kullback-Leibler divergence, we perform experiments by simulations. The Bhattacharyya distance is defined as [1]

$$B(I_x, I_y) = \sqrt{1 - \rho(p_x, p_y)}, \tag{4.13}$$

where the Bhattacharyya coefficient ρ is given by

$$\rho(p_x, p_y) = \sum_{u=1}^{m} \sqrt{\hat{p}_x(u)\hat{p}_y(u)}. \tag{4.14}$$

The Kullback-Leibler divergence between two distribution is defined as

$$D(I_x, I_y) = \int p_y(u) \log \frac{p_y(u)}{p_x(u)} du. \tag{4.15}$$

We first generate two multivariate Gaussian distributions

$$\begin{aligned} p_x(u) &\sim G(\mu_1, I) \\ p_y(u) &\sim G(\mu_2, I) \end{aligned}$$

where $\mu_1 = (\mu, 0, \ldots, 0)$, $\mu_2 = -\mu_1$, μ varies from 0 to 1.5, and I is an identity covariance matrix. For dimensions 3, 5 and 7, 100 samples were generated for each distribution and 100 repetitions were run. The estimated distances between two distributions *w.r.t.* the ground truth are displayed in Figure 4.1. We also generate

[1] We use the definition in [24], which is slightly different from the one defined in [122]. In order to be consistent with the previous literature, we follow the terminology "distance" instead of more accurate one "measure" as in [122].

two distributions in dimensions between 1 and 7 (the histogram based methods run out of memory beyond dimension 7), the centers are located at $\mu_1 = (1, 1, \ldots, 1)$ and $\mu_2 = -\mu_1$. The estimated distances between two distributions *w.r.t.* the ground truth are displayed in Figure 4.2. The simulations indicate that the Bhattacharyya distance and Kullback-Leibler divergence are incompetent in higher dimensions and the computations in higher dimensions are instable. In contrast, our similarity measure are much better and stable in both lower and higher dimensions. The intuition behind this phenomenon is so called "small sample effect" [33], because the data population in tracking applications is in general very small in higher dimensions.

4.3 Similarity-Based Tracking Algorithms

The simple similarity measure (4.8) contains rich information about the correlation between the model image and the target image. The asymptotic behaviors have been discussed and the corresponding algorithms can be found in [85, 3, 11]. In this section, we will derive some tracking algorithms which represent an intermediate stage between histogram-based tracking and template-based methods.

4.3.1 Pure Translation

In the case of pure translation, the warps $W(\mathbf{x}; \mathbf{p})$ is

$$\mathbf{y} = W(\mathbf{x}; \mathbf{p}) = \mathbf{x} + \mathbf{p} = \begin{pmatrix} x_1 + p_1 \\ \\ x_2 + p_2 \end{pmatrix}, \tag{4.16}$$

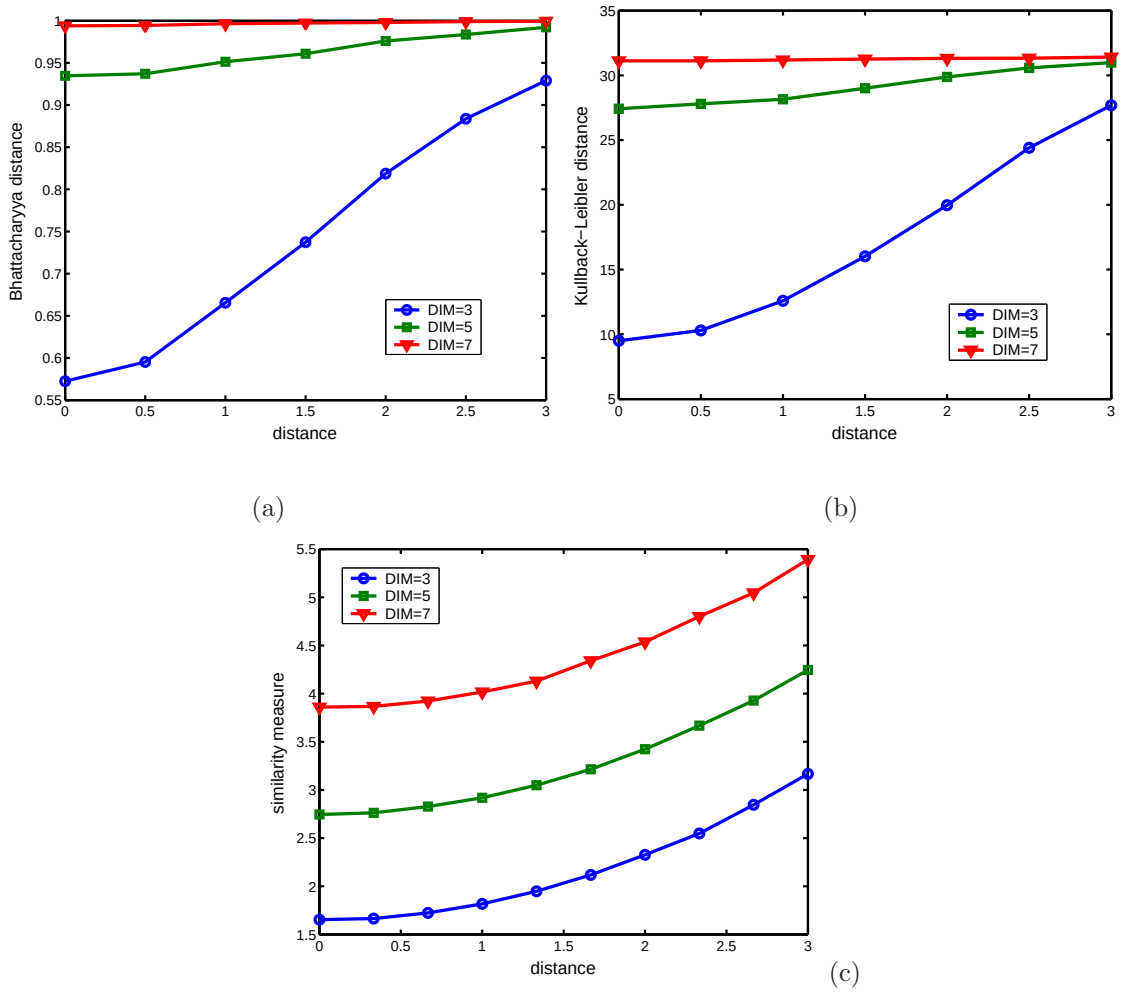
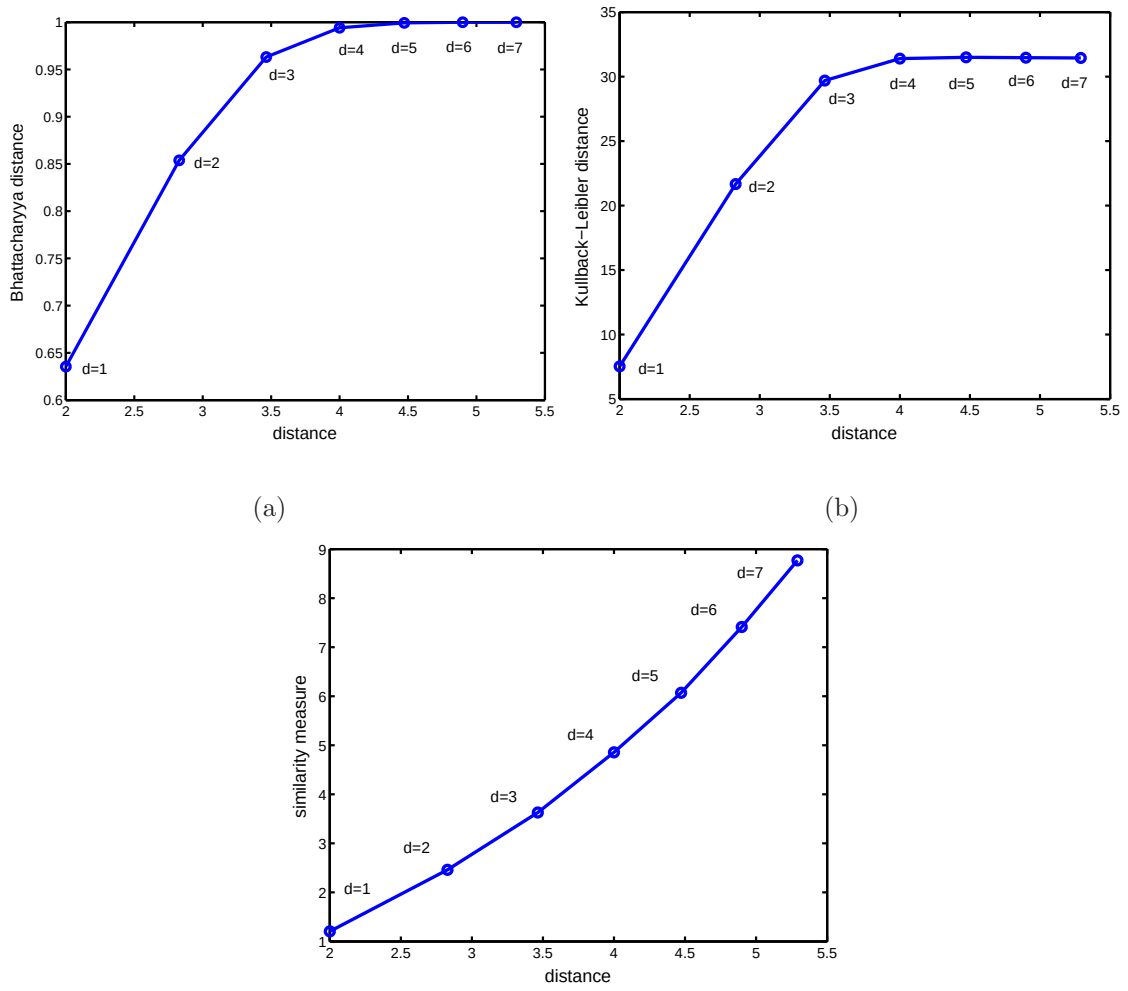


Figure 4.1: The estimated similarity measures between two distributions using: (a) Bhattacharyya distance, (b) Kullback-Leibler divergence, and (c) our similarity measure, *w.r.t.* the ground truth. The simulations are repeated 100 times for dimensions 3, 5 and 7, where the distances between the centers of two Gaussian distributions vary from 0 to 3. All simulations use an identity covariance matrix.



(a) (b)

(c)

Figure 4.2: The estimated similarity measures between two distributions using: (a) Bhattacharyya distance, (b) Kullback-Leibler divergence, and (c) our similarity measure, *w.r.t.* the ground truth. The simulations are repeated 100 times for each dimension between 1 and 7, where the centers of the Gaussian distributions are located at $(1, 1, \ldots, 1)$ and $(-1, -1, \ldots, -1)$. All simulations use an identity covariance matrix.

then the similarity measure (4.8) becomes

$$J(I_x, I_y) = \frac{1}{MN} \sum_{i=1}^{N} \sum_{j=1}^{M} w(\left\| \frac{\mathbf{y}_j - \mathbf{p} - \mathbf{x}_i}{\sigma} \right\|^2) k(\left\| \frac{\mathbf{u}_i - \mathbf{v}_j}{h} \right\|^2). \tag{4.17}$$

Let $\mathbf{x}_*$ be the center of model image and $\mathbf{y}$ be the center of target image, then $\mathbf{y} = \mathbf{x}_* + \mathbf{p}$, and equation (4.17) becomes

$$J(I_x, I_y) = \frac{1}{MN} \sum_{i=1}^{N} \sum_{j=1}^{M} w(\left\| \frac{(\mathbf{y}_j - \mathbf{y}) - (\mathbf{x}_i - \mathbf{x}_*)}{\sigma} \right\|^2) k(\left\| \frac{\mathbf{u}_i - \mathbf{v}_j}{h} \right\|^2). \tag{4.18}$$

Once we have the similarity measure between the model image and target image, we can find the target location in the target image by maximizing the similarity measure (4.8) or equivalently minimizing the distance (4.12) with respect to the variable $\mathbf{p}$, or equivalently to the variable $\mathbf{y}$. There are many techniques for searching for the optimal solution. Since the similarity function (4.8) is smooth and differentiable, and the displacement between the successive frames is small, we adopt the mean-shift algorithm [22] which has already proved successful in many computer vision applications [22, 24].

The gradient of the distance function (4.12) with respect to the vector $\mathbf{y}$ is

$$\nabla L(\mathbf{y}) = -\frac{\nabla J(\mathbf{y})}{J(\mathbf{y})}, \tag{4.19}$$

where

$$\nabla J(\mathbf{y}) = \frac{2}{MN\sigma^2} \sum_{i=1}^{N} \sum_{j=1}^{M} (\Delta\mathbf{x}_i - \Delta\mathbf{y}_j)\, k_{ij} w'(\left\| \frac{\Delta\mathbf{x}_i - \Delta\mathbf{y}_j}{\sigma} \right\|^2), \tag{4.20}$$

and $k_{ij} = k(\left\| \frac{\mathbf{u}_i - \mathbf{v}_j}{h} \right\|^2)$, $\Delta\mathbf{x}_i = \mathbf{x}_i - \mathbf{x}_*$, $\Delta\mathbf{y}_j = \mathbf{y}_j - \mathbf{y}$.

The *mean shift* of the smoothed similarity function $J(\mathbf{y})$ is

$$\nabla L(\mathbf{y}) = \frac{\sum_{i=1}^{N} \sum_{j=1}^{M} (\mathbf{y}_j - \mathbf{x}_i) k_{ij} g(\left\| \frac{\Delta\mathbf{x}_i - \Delta\mathbf{y}_j}{\sigma} \right\|^2)}{\sum_{i=1}^{N} \sum_{j=1}^{M} k_{ij} g(\left\| \frac{\Delta\mathbf{x}_i - \Delta\mathbf{y}_j}{\sigma} \right\|^2)} - \mathbf{y} + \mathbf{x}_*, \tag{4.21}$$

where $g(x) = -w'(x)$ is also the profile of a RBF kernel.

Given the sample points $\{\mathbf{x}_i, \mathbf{u}_i\}_{i=1}^N$ centered at $\mathbf{x}_*$ in the model image, and $\{\mathbf{y}_j, \mathbf{v}_j\}_{j=1}^M$ centered at the current position $\hat{\mathbf{y}}_0$ in the current target image, the object tracking based on the mean-shift algorithm is an iterative procedure which recursively moves the current position $\hat{\mathbf{y}}_0$ to the new position $\hat{\mathbf{y}}_1$ until reaching the density mode according to

$$\hat{\mathbf{y}}_1 = \frac{\sum_{i=1}^N \sum_{j=1}^M \mathbf{y}_j k_{ij} g_{ij}}{\sum_{i=1}^N \sum_{j=1}^M k_{ij} g_{ij}} - \frac{\sum_{i=1}^N \sum_{j=1}^M \mathbf{x}_i k_{ij} g_{ij}}{\sum_{i=1}^N \sum_{j=1}^M k_{ij} g_{ij}} + \mathbf{x}_*. \tag{4.22}$$

where $g_{ij} = g(\left\| \frac{\Delta\mathbf{x}_i - \Delta\mathbf{y}_j}{\sigma} \right\|^2)$.

In equation (4.22), the first term is the weighted centroid of the sample points $\{\mathbf{y}_j\}_{j=1}^M$ in the target image, which we denote as $\mathbf{c}_\mathbf{y}$. The second term is the weighted centroid of the sample points $\{\mathbf{x}_i\}_{i=1}^N$ in the model image, which we denote as $\mathbf{c}_\mathbf{x}$. The weight from the kernel function $k(x)$ encourages pairs of similar vectors in feature space and penalizes mismatched pairs. The weight from kernel function $g(x)$ enforces the spatial correlations. Since the kernel functions we used are convex and smooth RBFs, it can be proved that the above mean-shift procedure converges and that the similarity measure (4.8) monotonically increases as in [24].

As shown in Figure 4.3, if the tracked shape is symmetric, we have

$$\mathbf{c}_\mathbf{y} - \mathbf{y} = -(\mathbf{c}_\mathbf{x} - \mathbf{x}_*). \tag{4.23}$$

In general case, the two displacements in equation (4.23) are approximately equal, then we have the following updating rule

$$\hat{\mathbf{y}}_1 = \hat{\mathbf{y}}_0 + 2(\mathbf{c}_\mathbf{y} - \hat{\mathbf{y}}_0) = 2\mathbf{c}_\mathbf{y} - \hat{\mathbf{y}}_0. \tag{4.24}$$

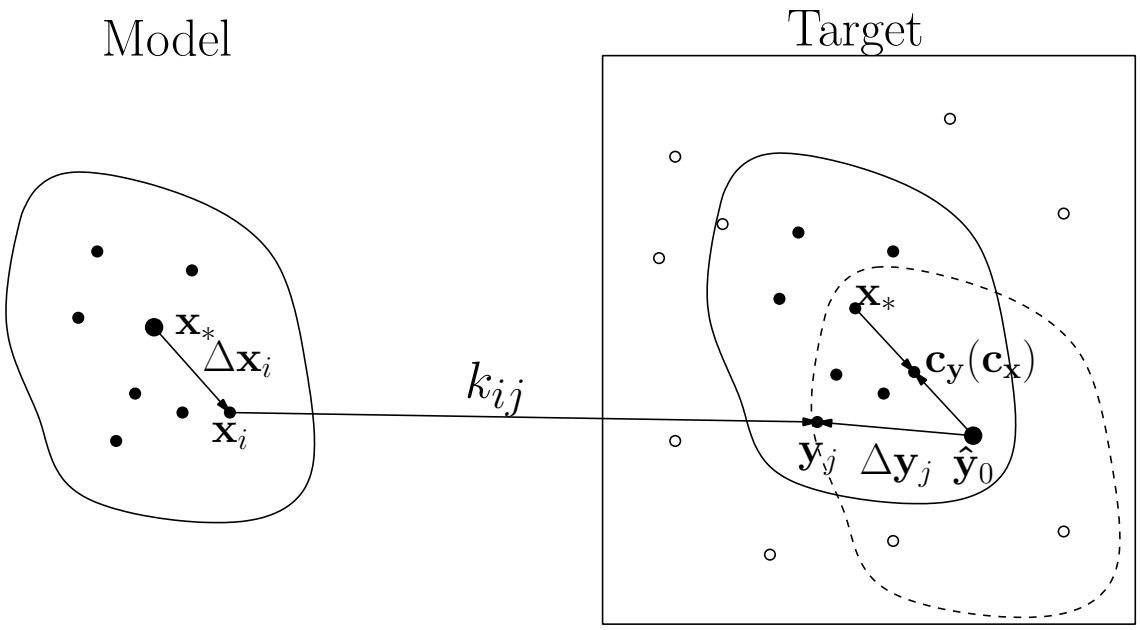


Figure 4.3: The mean-shift based tracking procedure. At each step of the mean-shift procedure, the new location of the target is a combination of the weighted centroids of the points within the current region (solid line) and the old region (dashed line). The weight is a combination of k_{ij} and g_{ij}.

In the literature [23, 20, 24, 19], the current location of the target is the weighted centroid of the samples in the target image. As shown in Figure 4.3, the weighted centroid is midway to the correct location of the target.

4.3.2 Translation and Scaling

If the target only performs pure translation in the image sequence, the above algorithm gives very good results. However, it is not uncommon in practice that the size of the target changes between frames. Here we can model the motion as

translation plus scaling, the similarity measure becomes:

$$J(I_x, I_y) = \frac{1}{MN}\sum_{i=1}^{N}\sum_{j=1}^{M} w\left(\left|\frac{\sqrt{s}\Delta\mathbf{x}_i}{\sigma} - \frac{\Delta\mathbf{y}_j}{\sqrt{s}\sigma}\right|^2\right) k\left(\left|\frac{\mathbf{u}_i - \mathbf{v}_j}{h}\right|^2\right), \tag{4.25}$$

where s is the scaling factor accounting for the size changes of target between frames [99].

Similarly, we have the *mean shift* vector for $\mathbf{y}$ is

$$\nabla_{\mathbf{y}} L(\mathbf{y}, s) = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M}(\mathbf{y}_j - \frac{s}{\sigma}\mathbf{x}_i)k_{ij}g_{ij}}{\sum_{i=1}^{N}\sum_{j=1}^{M} k_{ij}g_{ij}} - \mathbf{y} + \frac{s}{\sigma}\mathbf{x}_*, \tag{4.26}$$

where $g_{ij} = -w'(\left\|\frac{\Delta\mathbf{x}_i}{\sigma} - \frac{\Delta\mathbf{y}_j}{s}\right\|^2)$ is the profile of a RBF kernel. The updating rule for $\mathbf{y}$ is

$$\hat{\mathbf{y}}_1 = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M}\mathbf{y}_j k_{ij}g_{ij}}{\sum_{i=1}^{N}\sum_{j=1}^{M} k_{ij}g_{ij}} - \frac{s\sum_{i=1}^{N}\sum_{j=1}^{M}\mathbf{x}_i k_{ij}g_{ij}}{\sigma\sum_{i=1}^{N}\sum_{j=1}^{M} k_{ij}g_{ij}} + \frac{s}{\sigma}\mathbf{x}_*. \tag{4.27}$$

The *mean shift* vector for scaling factor s is

$$\nabla_s L(\mathbf{y}, s) = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M}(\|\frac{\Delta\mathbf{y}_j}{s}\|^2 - \|\frac{\Delta\mathbf{x}_i}{\sigma}\|^2)k_{ij}g_{ij}s}{\sum_{i=1}^{N}\sum_{j=1}^{M} k_{ij}g_{ij}}, \tag{4.28}$$

The *mean shift* vector (4.28) is similar to the one derived using scale space theory in [19] if we use the Gaussian kernel functions. But it also contains the second order moment information which is used in [15] for estimating the scaling factor.

The updating rule for scaling factor s is

$$\hat{s}_1 = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M}(1 + \|\frac{\Delta\mathbf{y}_j}{\hat{s}_0}\|^2 - \|\frac{\Delta\mathbf{x}_i}{\sigma}\|^2)k_{ij}g_{ij}}{\sum_{i=1}^{N}\sum_{j=1}^{M} k_{ij}g_{ij}}\hat{s}_0, \tag{4.29}$$

Since the scaling factor s is always positive, the above updating rule must guarantee to generate a positive new scaling $\hat{s}_1$ given current positive $\hat{s}_0$.

4.3.3 General Motion

In the above cases, we obtained a robust tracking algorithm with the pure translation and scaling. There is no need for explicitly computing spatial correspondence between pixels in the target and the model images. On the other hand, the spatial constraints are lost and only the position of the targets has been tracked. In this section, we will derive a tracking algorithm with the general geometric transformation $\mathbf{y} = \mathbf{W}(\mathbf{x}; \mathbf{p})$.

The gradient of the distance function (4.8) with respect to the vector $\mathbf{p}$ is

$$\begin{aligned}
\nabla J(\mathbf{p}) &= \mathbf{G}_1(\mathbf{p}) + \mathbf{G}_2(\mathbf{p}) \\
&= \frac{2}{MNh^2} \sum_{i=1}^{N} \sum_{j=1}^{M} k' \left(\| \frac{\mathbf{v}_j - \mathbf{u}_i}{h} \|^2 \right) w \left(\| \frac{\mathbf{y}_j - \mathbf{x}_i}{\sigma} \|^2 \right) \left[\nabla \mathbf{v}_j \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T (\mathbf{v}_j - \mathbf{u}_i) \\
&+ \frac{2}{MN\sigma^2} \sum_{i=1}^{N} \sum_{j=1}^{M} w' \left(\| \frac{\mathbf{y}_j - \mathbf{x}_i}{\sigma} \|^2 \right) k \left(\| \frac{\mathbf{v}_j - \mathbf{u}_i}{h} \|^2 \right) \left[\frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^T (\mathbf{y}_j - \mathbf{x}_i)
\end{aligned} \tag{4.30}$$

where $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ is the *Jacobian* of the warp:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots & \frac{\partial W_x}{\partial p_n} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots & \frac{\partial W_y}{\partial p_n} \end{pmatrix} \tag{4.31}$$

and $\nabla \mathbf{v}_j$ is the image *gradient* of the target image at pixel j.

In equation (4.30), the first term $\mathbf{G}_1(\mathbf{p})$ is counterpart of the gradient in the Lucas-Kanade algorithm [85, 3] which contributes to the template matching. The Lucas-Kanade algorithm leads to an *iteratively reweighted least squares algorithm*, if robust error function is adopted [3]. The pixels with large residual will get smaller weights to eliminate the effect of outlier. However the large residual is not necessarily the outliers. One example is the pixels on the edges, which carry more

information about the correlation. In our scheme, the first term is similar to the *robust anisotropic diffusion* which preserves the edges at the same time filtering the error images [12, 115].

The second term $\mathbf{G}_2(\mathbf{p})$ is the counterpart of equation (4.20) which accounts for recovering the position of the target. The *steepest descent* step of expression (4.8) is

$$\Delta p = -\frac{\nabla J(\mathbf{p})}{J(\mathbf{p})}. \tag{4.32}$$

The whole algorithm is as follows:

Algorithm 1 The two-stage tracking algorithm

repeat

(1) Compute $\Delta\mathbf{p} = -\mathbf{G}_2/J$ with warping $\mathbf{W}(\mathbf{x};\mathbf{p})$

(2) Update $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$

until $\|\Delta\mathbf{p}\| \leq \epsilon_1$

repeat

(3) Compute $\Delta\mathbf{p} = -\nabla J/J$ with warping $\mathbf{W}(\mathbf{x};\mathbf{p})$

(4) Update $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$

until $\|\Delta\mathbf{p}\| \leq \epsilon$

The benefit of the above two-stage algorithm is that the position parameters of the target are first estimated which will facilitate the second stage template matching and greatly improve the robustness and accuracy of tracking. To improve the convergence rate of the algorithm we can apply the Gauss-Newton method in the second stage as in [3].

4.4 Speedup using the Improved FGT

The computational complexity per frame in the above algorithm is $O(PMN)$, where P is the average number of iterations per frame, M and N are the number of sample points in target image and model image respectively. Typically the average number of iterations per frame P is less than ten and $M \approx N$. Then the order of the computational complexity is quadratic. While the above simple algorithm runs at real-time frame rate when the number of points N is small, say up to 100, it will slow down quadratically with the number of sample points.

From now on, we use the Gaussian kernel (4.2) in the above tracking algorithm. We apply the fast Gauss transform (FGT) [57, 133] to the tracking algorithm to reduce its computational complexity from quadratic order to linear order.

Since the derivative of the Gaussian kernel is still a Gaussian, the mean shift based object tracking with the Gaussian kernel is

$$\hat{\mathbf{y}}_1 = \frac{2\sum_{j=1}^{M} \mathbf{y}_j f(\mathbf{y}_j)}{\sum_{j=1}^{M} f(\mathbf{y}_j)} - \hat{\mathbf{y}}_0, \tag{4.33}$$

where

$$f(\mathbf{y}_j) = \sum_{i=1}^{N} e^{-\|\mathbf{u}_i - \mathbf{v}_j\|^2/h^2} e^{-\|(\mathbf{y}_j - \hat{\mathbf{y}}_0) - (\mathbf{x}_i - \hat{\mathbf{x}}_*)\|^2/\sigma^2} \tag{4.34}$$

is a *discrete Gauss transform* of $\mathbf{y}_j$ for $j = 1, \ldots, M$. The vectors $\mathbf{u}_i$ are called "sources" and $\mathbf{v}_j$ are called "targets".

The computational complexity of a direct evaluation of the discrete Gauss transform (4.34) requires $O(MN)$ operations. In low-dimensional spaces, the computational complexity has been reduced by Greengard and Strain [57] to $C \cdot (M+N)$

using the fast Gauss transform, where constant factor C depends on the precision required and dimensionality.

The fast Gauss transform is based on a divide-and-conquer strategy. The source points are subdivided into uniform boxes. The contributions from the sources are collected to the centers of the boxes by means of Hermite expansions and Taylor series. Then the contributions are distributed to each target point from the box centers by consolidating the expansions at each target point.

Although the fast Gauss transform achieved great success in low dimensions, the performance in higher dimensions is poor. The reason is that the fast Gauss transform is originally designed for solving the heat equation whose dimension is up to three. There are two major drawbacks in the original FGT. One is that the number of boxes in FGT grows exponentially with dimensionality. The other is that the number of terms in the expansions grows exponentially with the dimensionality, too. So the performance of the FGT degrades exponentially with the dimensionality. We use the improved fast Gauss transform to deal with the above serious drawbacks of the FGT in higher dimensions and achieve real-time performance for the object tracking.

4.5 Experimental Results

In this section, we present some real-time object tracking results using the proposed algorithm. In the first two experiments, the RGB color space is used as the feature space, and in the third one, the RGB color space plus 2D image gradient

is used. The 2D spatial domain is combined to the feature space. The Gaussian kernel (4.2) is used in all the experiments. The algorithm is implemented in C++ with Matlab interface and runs on a 900MHZ PIII PC.

We first compare results on 2 clips that were used in [24]. The first clip is the *Football* sequence which has 154 frames of size 352×240. The tracking algorithm is initialized with a manually selected region in frame 0 of size 60×60. The bandwidth in the feature space is $h = 20$ and in the spatial domain is $\sigma = 10$. The algorithm tracks the player reliably with partial occlusion, clutter, blurring and compression noise (see Figure 4.4). The number of mean-shift iterations is shown in Figure 4.5. The average number of the iterations is 2.3179 per frame and the average processing time per frame: $0.0291s$. The number of iterations required in each frame for this sequence are shown in Figure 4.5. The number of iterations required in [24] for each corresponding frame (see Figure 2 in [24]) is larger, where the average number of iterations is 4.19 per frame. This shows that our similarity measure (4.8) functions is as good or better than the Bhattacharyya coefficient used in [24].

The second experiment uses the *Ball* sequence. If we blindly apply the tracking algorithm, it will either track the background if a large region is used, or lose the ball if the tracking region is small and the movement is large. Another advantage of experimenting with such a sequence is that the target (the ball) is relatively simpler and easier to make comparisons. We utilize the background information and assume a mask about the tracked object is available. We initialize the model with a region in frame 3 size of 48×48. The bandwidths are $(h, \sigma) = (18, 12)$. We only keep the foreground pixels in the model and run the algorithm as in the previous exper-

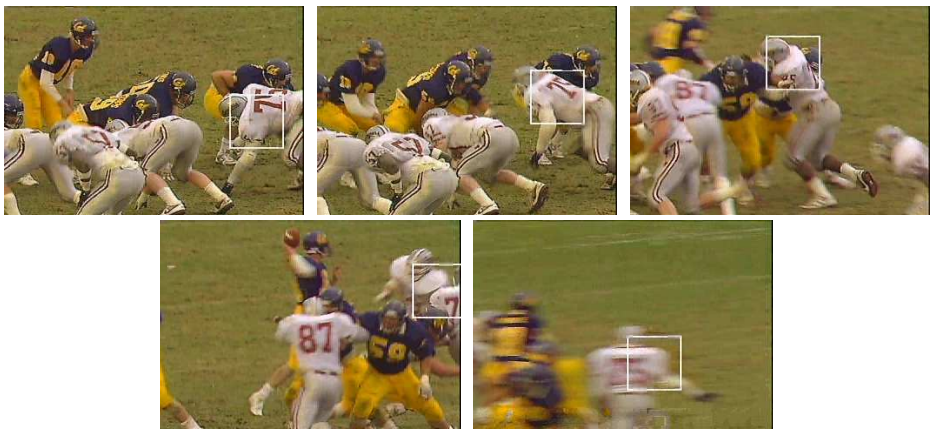


Figure 4.4: Tracking results of the *Football* sequence. Frames 30, 75, 105, 140 and 150 are displayed.

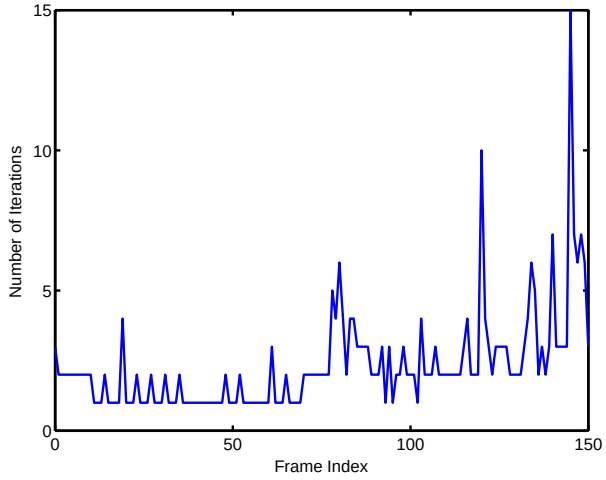
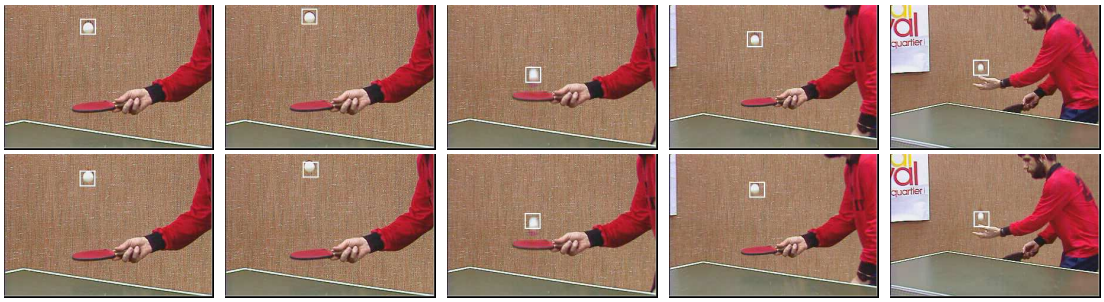


Figure 4.5: The number of mean-shift iterations *w.r.t.* the frame index for the *Football* sequence.



Frame 6 Frames 20 Frame 27 Frame 36 Frame 53

Figure 4.6: Tracking results of the *Ball* sequence using (top row) our similarity measure and (second row) Bhattacharyya distance.

iment. The algorithm reliably and accurately tracks the ball with average number of iterations 2.7679 and average processing time per frame $0.0169s$. In contrast, to successfully track this sequence, in [24] a background-weighted histogram was employed. The tracking results shown in Figure 4.6 are more accurate than those in [24]. The number of iterations and sums of squared differences between the model image and the tracked images are shown in Figure 4.7. The results of our method are more accurate and number of iterations is smaller than the method using the Bhattacharyya distance.

If more features are available, we can conveniently integrate the feature information into high dimensional feature-spatial spaces. In the third experiment a more complex clip is taken. In order to track a face with changing appearance and complex background, we use both the RGB color space and 2D image gradients as features. The image gradients are the horizontal and vertical image gradients of the grayscale image obtained using the Sobel operator. We initialize the model with a

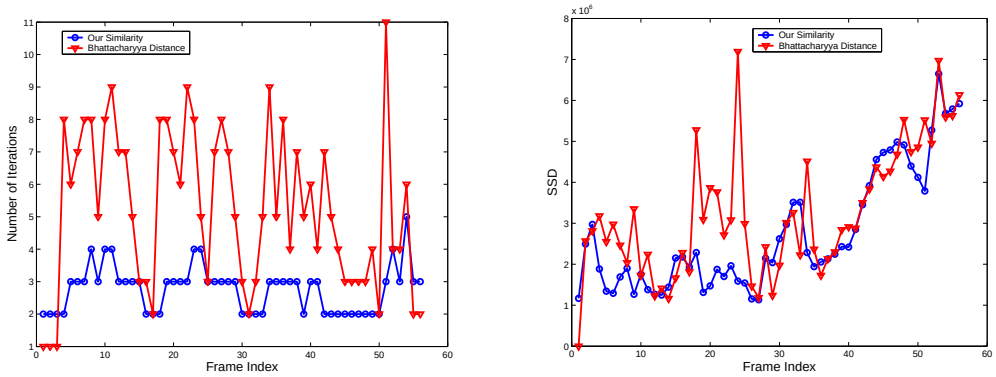


Figure 4.7: The number of iterations (*left*) and sums of squared differences (*right*) *w.r.t.* the frame index for the *Ball* sequence using our similarity measure and Bhattacharyya distance.

region in frame 0 size of 24×24. The bandwidths are $(h, \sigma) = (25, 12)$. The average number of iteration per frame is 2.1414 and average processing time per frame is $0.0044s$. The algorithm reliably tracks the face and results are shown in Figure 4.8.

In the fourth experiment, we will study the convergence rate with respect to the step size. We select a 100×100 region from a face image as the model image. Then we shift this region by 40 pixel as shown in Figure 4.9. From the results, we find that the tracking algorithm using double step size and triple step size accelerates the convergence rate. But the tracking algorithm using triple step size is not as stable and accurate as the double step size.

The fifth experiment is to test the two-stage algorithm. We crop a 100×100 region from a face image and warp it into a model image by a randomly generated affine transformation. The affine transformation is generated as in [3]: three canonical points $(0, 0)$, $(99, 0)$ and $(49, 99)$ are randomly perturbed and translated

Figure 4.8: Tracking results of the *Walking* sequence. Frames 4, 19, 50, 99, 166 and 187 are displayed.

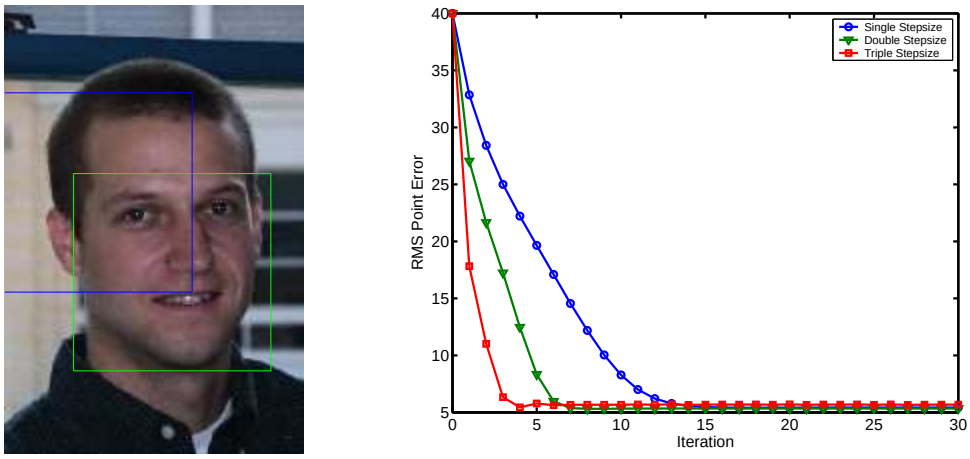
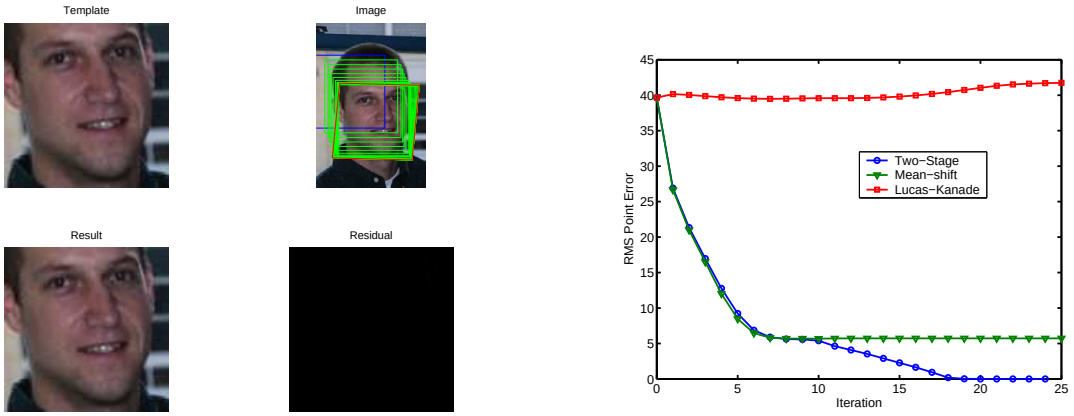


Figure 4.9: Object tracking using different step sizes. *Left*: The face image and the template region (inside the green frame), the starting position (blue frame). *Right*: RMS point error *w.r.t.* the iteration. Blue curve for single stepsize, green curve for double stepsize and red curve for triple stepsize.

with additive Gaussian noise, then fitted with the affine transformation. We start the tracking algorithms from the identity transformation. The RMS error is measured on the three canonical points between their current and correct locations. The results of the three algorithms are displayed in Figure 4.10. We can find that the Lucas-Kanade algorithm diverges. The mean-shift based tracking using feature space approaches the correct location rapidly in the beginning iterations but cannot accurately fit the model image. The two-stage algorithm converges to the correct warping accurately and rapidly.

We also test the algorithms on the homography transformation. The canonical points are the 4 corners of the model image. The results of the three algorithms are displayed in Figure 4.11. Same conclusion as the case of affine transformation can



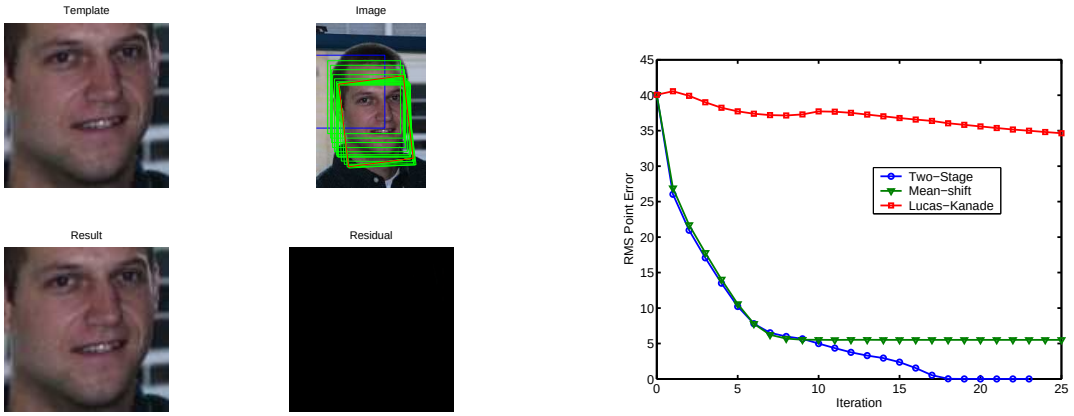
(a) (b)

Figure 4.10: Tracking results of three algorithm on the affine transformation. (a) The result of our two-stage algorithm. (b) The RMS point errors of the three algorithms.

be drawn.

4.6 Discussion and Conclusions

In this chapter we proposed a novel simple symmetric similarity function between spatially-smoothed kernel-density estimates of the model and target distributions for object tracking. The similarity measure is based on the expectation of the density estimates over the model or target image. The well-known radial-basis kernel functions are used to measure the affinity between points and provide a better outlier rejection property. To track the objects, the similarity function is maximized using the mean-shift algorithm to iteratively find the local mode of the function. The tracking algorithm based on this similarity function is very simple and we attach the actual Matlab code for tracking in the Appendix (without the fast Gauss



(a) (b)

Figure 4.11: Tracking results of three algorithm on the homography transformation. (a) The result of our two-stage algorithm. (b) The RMS point errors of the three algorithms.

transform). In case of general motion, we developed a two-stage algorithm which relies on the feature space to robustly recover the position of the target and utilizes the spatial template to fit the target accurately.

Since the similarity measure is an average taken over all pairs of the pixel between two distributions, the computational complexity is quadratic. To alleviate the quadratic complexity, we employ Gaussian kernels and the fast Gauss transform to reduce the computations to linear order. This leads to a very efficient and robust nonparametric tracking algorithm. It also very convenient for integration of the background information and generalization to high dimensional feature space. The similarity is directly based on the kernel density estimation, there are no stability and singularity problems which perplex the information-theory based distance measures. In this chapter we use a fixed bandwidth which by no means is optimal for the

performance. The variable and adaptive bandwidth selection will be studied in the future work.

Appendix: Matlab Code for Tracking

Attached below is actual Matlab code that implements the tracking algorithm with the similarity function (4.8). Note that this Matlab code does not include the improved fast Gauss transform (IFGT). This code achieved tracking speeds of about 2.5s per frame for a region of size 12×12. With the inclusion of the IFGT the tracking speeds are substantially faster.

```
function  [newpos, nits] = mspos(initimg, newimg, sig, h,...
                            initpos, oldpos, epsilon, maxits)
%MSPOS Mean-shift based object tracking algorithm
%
% INPUT:
%   initimg : Model or template image
%   newimg  : Image of current frame
%   sig     : Bandwidth in the feature space
%   h       : Bandwidth in the spatial space
%   initpos : Position of the target in model image
%   oldpos  : Current position of the target
%   epsilon : Termination criterion
%   maxits  : Maximum number of iterations allowed
% OUTPUT:
%   newpos  : Predicted position of the target
%   nits    : Number of iterations used
```

```
%
%   Copyright 2003 by Changjiang Yang.
%   $Revision: 1.3 $  $Date: Mon Mar 29 09:19:55 EST 2004 $

% Call function inddisk to find the pixels inside of a circle
[ix1,ix2] = inddisk(initimg,initpos,sig);
sig2 = 2*sig*sig; h2 = 2*h*h;
y = oldpos;
% Iteratively find the correct position using mean-shift algorithm
for k = 1:maxits,
    % Find all pixels on current frame inside of a circle centered at y
    [jy1,jy2] = inddisk(newimg,y,sig);
    y0 = y; sumxyuv = 0.0; sumyxyuv = zeros(size(y));
    % Compute the similarity measure using double loop
    for i = 1:length(ix1),
        % Find all pixels inside of a circle on template image
        ix = [ix1(i) ix2(i)];
        dx = initpos - ix;
        ui = initimg(ix2(i),ix1(i),:); ui = ui(:);
        % Inside loop
        for j = 1:length(jy1),
            jy = [jy1(j) jy2(j)]; dy = y - jy;
            vj = newimg(jy2(j),jy1(j),:); vj = vj(:);
            duv = ui - vj; duv2 = duv.'*duv;
            dxy = dx - dy; dxy2 = dxy*dxy';
            % Compute the weights for all pixels inside of a circle
            wt = exp(-(dxy2/sig2 + duv2/h2));
```

```
            sumxyuv = sumxyuv + wt;
            % Compute the weighted sums of the positions
            sumyxyuv = sumyxyuv + wt*(jy-ix);
        end
    end
    % Next position is the weighted sum of the current pixel positions
    y = sumyxyuv / sumxyuv + initpos;
    % Check for convergence
    if norm(y - y0) < epsilon, break; end
end
newpos = y;
nits = min(k,maxits);
return;


%INDDISK Find all pixels inside of a circle
% INPUT:
%   img : Input image
%   pos : Center of the circle
%   h   : Radius of the circle
% OUTPUT:
%   ix  : Indices of x-coordinate
%   iy  : Indices of y-coordinate
function  [ix,iy] = inddisk(img,pos,h)
siz = size(img);
% Generate mesh grids using Matlab function meshgrid
[XX,YY] = meshgrid(1:siz(1),1:siz(2));
% Find all pixels inside of the circle
```

```
[ix,iy] = find((XX-pos(2)).^2 + (YY-pos(1)).^2 < h^2);

return;
```

Chapter 5

Efficient Kernel Machines Using the IFGT

5.1 Introduction

Kernel based methods, including support vector machines [117], regularization networks [51] and Gaussian processes [124], have attracted much attention in machine learning. The solid theoretical foundations and good practical performance of kernel methods make them very popular. However one major drawback of the kernel methods is their scalability. Kernel methods require $O(N^2)$ storage and $O(N^3)$ operations for direct methods, or $O(N^2)$ operations per iteration for iterative methods, which is impractical for large datasets.

To deal with this scalability problem, many approaches have been proposed, including the Nyström method [123], sparse greedy approximation [111, 110], low rank kernel approximation [44] and reduced support vector machines [79]. All these try to find a reduced subset of the original dataset using either random selection or greedy approximation. In these methods there is no guarantee on the approximation of the kernel matrix in a deterministic sense. An assumption made in these methods is that most eigenvalues of the kernel matrix are zero. This is not always true and its violation results in either performance degradation or negligible reduction in computational time or memory.

We explore a deterministic method to speed up kernel machines using the

improved fast Gauss transform (IFGT) [133, 134]. The kernel machine is solved iteratively using the conjugate gradient method, where the dominant computation is the matrix-vector product which we accelerate using the IFGT. Rather than approximating the kernel matrix by a low-rank representation, we approximate the matrix-vector product by the improved fast Gauss transform to any desired precision. The total computational and storage costs are of linear order in the size of the dataset. We present the application of the IFGT to kernel methods in the context of the Regularized Least-Squares Classification (RLSC) [101, 96], though the approach is general and can be extended to other kernel methods.

5.2 Regularized Least-Squares Classification

The RLSC algorithm [101, 96] solves the binary classification problems in Reproducing Kernel Hilbert Space (RKHS) [121]: given N training samples in d-dimensional space $\mathbf{x}_i \in \mathcal{R}^d$ and the labels $y_i \in \{-1, 1\}$, find $f \in \mathcal{H}$ that minimizes the regularized risk functional

$$\min_{f \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} V(y_i, f(\mathbf{x}_i)) + \lambda \|f\|_K^2, \tag{5.1}$$

where $\mathcal{H}$ is an RKHS with reproducing kernel K, V is a convex cost function and λ is the regularization parameter controlling the tradeoff between the cost and the smoothness. Based on the Representer Theorem [121], the solution has a representation as

$$f_\lambda(\mathbf{x}) = \sum_{i=1}^{N} c_i K(\mathbf{x}, \mathbf{x}_i). \tag{5.2}$$

If the loss function V is the hinge function, $V(y, f) = (1 - yf)_+$, where $(\tau)_+ = \tau$ for $\tau > 0$ and 0 otherwise, then the minimization of (5.1) leads to the popular Support Vector Machines which can be solved using quadratic programming.

If the loss function V is the square-loss function, $V(y, f) = (y - f)^2$, the minimization of (5.1) leads to the so-called Regularized Least-Squares Classification which requires only the solution of a linear system. The algorithm has been rediscovered several times and has many different names [101, 96, 50, 114]. In this chapter, we stick to the term "RLSC" for consistency. It has been shown in [101, 50] that RLSC achieves accuracy comparable to the popular SVMs for binary classification problems.

If we substitute (5.2) into (5.1), and denote $\mathbf{c} = [c_1, \ldots, c_N]^T$, $K = K(\mathbf{x}_i, \mathbf{x}_j)$, we can find the solution of (5.1) by solving the linear system

$$(K + \lambda' I)\mathbf{c} = \mathbf{y} \tag{5.3}$$

where $\lambda' = \lambda N$, I is the identity matrix, and $\mathbf{y} = [y_1, \ldots, y_N]^T$.

There are many choices for the kernel function K. The Gaussian is a good kernel for classification and is used in many applications. If a Gaussian kernel is applied, as shown in [96], the classification problem can be solved by the solution of a linear system, i.e., Regularized Least-Squares Classification. A direct solution of the linear system will require $O(N^3)$ computation and $O(N^2)$ storage, which is impractical even for problems of moderate size.

An effective way to solve the large-scale linear system (5.3) is to use iterative methods. Since the matrix K is symmetric, we consider the well-known *conjugate*

Algorithm 2 Regularized Least-Squares Classification

Require: Training dataset $S_N = (\mathbf{x}_i, y_i)_{i=1}^N$.

1. Choose the Gaussian kernel: $K(\mathbf{x}, \mathbf{x}') = e^{-\|\mathbf{x}-\mathbf{x}'\|^2/\sigma^2}$.

2. Find the solution as $f(\mathbf{x}) = \sum_{i=1}^N c_i K(\mathbf{x}, \mathbf{x}_i)$, where $\mathbf{c}$ satisfies the linear system (5.3).

gradient method. The conjugate gradient method solves the linear system (5.3) by iteratively performing the matrix-vector multiplication $K\mathbf{c}$. If $rank(K) = r$, then the conjugate gradient algorithm converges in at most $r+1$ steps. Only one matrix-vector multiplication and $10N$ arithmetic operations are required per iteration. Only four N-vectors are required for storage. So the computational complexity is $O(N^2)$ for low-rank K and the storage requirement is $O(N^2)$. While this represents an improvement for most problems, the rank of the matrix may not be small, and moreover the quadratic storage and computational complexity are still too high for large datasets. In the following sections, we present an algorithm to reduce the computational and storage complexity to linear order.

5.3 IFGT Accelerated RLSC: Discussion and Experiments

The matrix-vector product $K\mathbf{c}$ can be written in the form of the so-called *discrete Gauss transform* [57]. The key idea of all acceleration methods is to reduce the cost of the matrix-vector product. In reduced subset methods, this is performed by evaluating the product at a few points, assuming that the matrix is low rank. The general Fast Multipole Methods (FMM) seek to analytically approximate the

possibly full-rank matrix as a *sum* of low rank approximations with a tight error bound [113] (The FGT is a variant of the FMM with Gaussian kernel). It is expected that these methods can be more robust, while at the same time achieve significant acceleration.

The problems to which kernel methods are usually applied are in higher dimensions, though the intrinsic dimensionality of the data is expected to be much smaller. The original FGT does not scale well to higher dimensions. Its cost is of linear order in the number of samples, but exponential order in the number of dimensions. The improved FGT uses new data structures and a modified expansion to reduce this to polynomial order.

Despite this improvement, at first glance, even with the use of the IFGT, it is not clear if the reduction in complexity will be competitive with the other approaches proposed. Reason for hope is provided by the fact that in high dimensions we expect that the IFGT with very low order expansions will converge rapidly (because of the sharply vanishing exponential terms multiplying the expansion in factorization (2.11)). Thus we expect that combined with a dimensionality reduction technique, we can achieve very competitive solutions.

In this chapter we explore the application of the IFGT accelerated RLSC to certain standard problems that have already been solved by the other techniques. While dimensionality reduction would be desirable, here we do not perform such a reduction for fair comparison. We use small order expansions ($p = 1$ and $p = 2$) in the IFGT and run the iterative solver.

In the first experiment, we compared the performance of the IFGT on ap-

proximating the sums (4.34) with the Nyström method [123]. The experiments were carried out on a Pentium 4 1.4GHz PC with 512MB memory. We generate N source points and N target points in 100 dimensional unit hypercubes using a uniform distribution. The weights on the source points are generated using a uniform distribution in the interval $[0, 1]$. We directly evaluate the sums (4.34) as the ground truth, where $\sigma^2 = (0.5)d$ and d is the dimensionality of the data. Then we estimate it using the improved fast Gauss transform and Nyström method. To compare the results, we use the maximum relative error to measure the precision of the approximations. Given a precision of 0.5%, we use the error bound (2.23) to find the parameters of the IFGT, and use a trial and error method to find the parameter of the Nyström method. Then we vary the number of points, N, from 500 to 5000 and plot the time against N in Figure 5.1 (a). The results show the IFGT is much faster than the Nyström method. We also fix the number of points to $N = 1000$ and vary the size of centers (or random subset) k from 10 to 1000 and plot the results in Figure 5.1 (b). The results show that the errors of the IFGT are not sensitive to the number of the centers, which means we can use very a small number of centers to achieve a good approximation. The accuracy of the Nyström method catches up at large k, where the direct evaluation may be even faster. The intuition is that the use of expansions improves the accuracy of the approximation and relaxes the requirement of the centers.

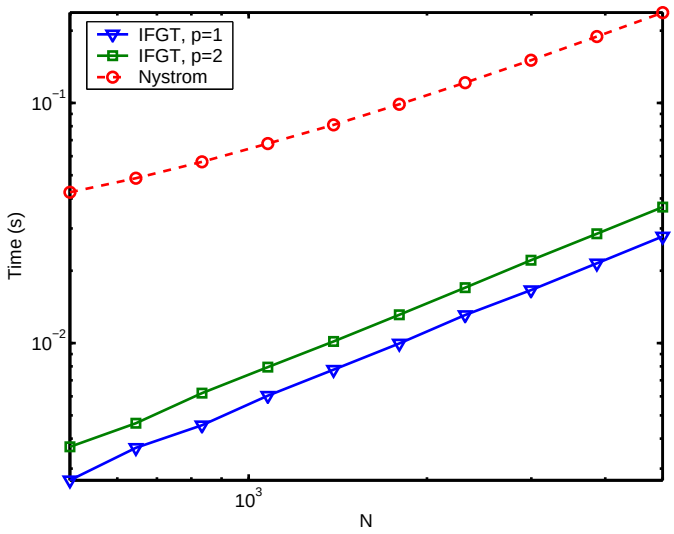
In the second experiment, five datasets from the UCI repository are used to compare the performance of four different methods for classification: RLSC with the IFGT, RLSC with full kernel evaluation, RLSC with the Nyström method and the

Proximal Support Vector Machines (PSVM) [50]. The Gaussian kernel is used for all these methods. We use the same value of $\sigma^2 = (0.5)d$ for a fair comparison. The ten-fold cross validation accuracy on training and testing and the training time are listed in Table 5.1. The RLSC with the IFGT is fastest among the four classifiers on all five datasets, while the training and testing accuracy is close to the accuracy of the RLSC with full kernel evaluation. The RLSC with the Nyström approximation is nearly as fast, but the accuracy is lower than the other methods. Worst of all, the Nyström approximation is numerically unstable, which results in the failure on the *Mushroom* dataset. The PSVM is accurate on the training and testing, but slow and memory demanding for large datasets, even with subset reduction.

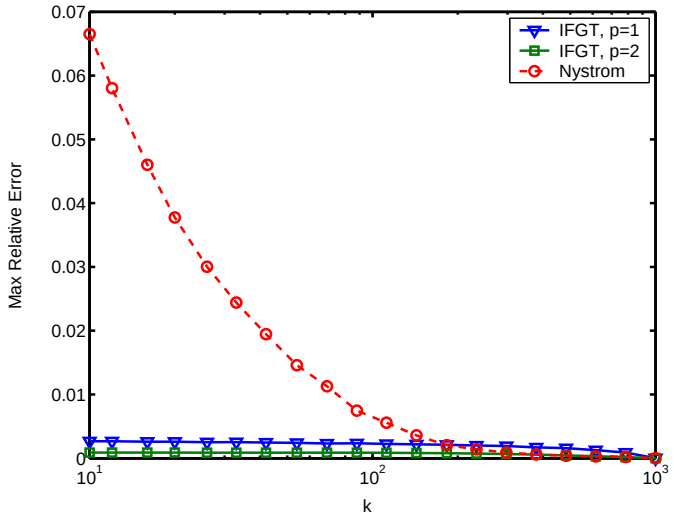
5.4 Conclusions and Discussion

We presented an improved fast Gauss transform to speed up kernel machines with Gaussian kernel to linear order. The simulations and the classification experiments show that the algorithm is in general faster and more accurate than other matrix approximation methods. At present, we do not consider the reduction from the support vector set or dimensionality reduction. The combination of the improved fast Gauss transform with these techniques should bring even more reduction in computation. Another improvement to the algorithm is an automatic procedure to tune the parameters. A possible solution could be running a series of testing problems and tuning the parameters accordingly. If the bandwidth is very small compared with the data range, the nearest neighbor searching algorithms could be

a better solution to these problems.



(a)



(b)

Figure 5.1: Performance comparison between the approximation methods. (a) Running time against N and (b) maximum relative error against k for fixed $N = 1000$ in 100 dimensions.

Table 5.1: Ten-fold training and testing accuracy in percentage and training time in seconds using the four classifiers on the five UCI datasets. Same value of $\sigma^2 = (0.5)d$ is used in all the classifiers. A rectangular kernel matrix with random subset size of 20% of N was used in PSVM on *Galaxy Dim* and *Mushroom* datasets.

Dataset Size × Dimension	RLSC+FGT %, %, s	RLSC %, %, s	Nyström %, %, s	PSVM %, %, s
Ionosphere	94.8400	97.7209	91.8656	95.1250
251 × 34	91.7302	90.6032	88.8889	94.0079
	0.3711	1.1673	0.4096	0.8862
BUPA Liver	79.6789	81.7318	76.7488	75.8134
345 × 6	71.0336	67.8403	69.2857	71.4874
	0.1279	0.4833	0.1475	0.3468
Tic-Tac-Toe	88.7263	88.6917	88.4945	92.9715
958 × 9	86.9507	85.4890	84.1272	87.2680
	0.3476	2.9676	1.8326	3.9891
Galaxy Dim	93.2967	93.3206	93.7023	93.6705
4192 × 14	93.2014	93.2258	93.7020	93.5589
	2.0972	78.3526	3.1081	44.5143
Mushroom	88.2556	87.9001		85.5955
8124 × 22	87.9615	87.6658	failed	85.4629
	14.7422	341.7148		285.1126

Chapter 6

Multiple Object Tracking

6.1 Introduction

As we previously stated, object tracking is an important task in many computer vision applications including surveillance, gesture recognition, smart rooms, vehicle tracking, augmented reality, video compression, and medical imaging, etc. The tracking of real-world objects is a challenging problem due to the presence of noise, occlusion, clutter and dynamic changes in the scene other than the motion of objects of interest. A variety of tracking algorithms have been proposed and implemented to overcome these difficulties; they can be roughly classified into two categories: deterministic methods and stochastic methods.

Deterministic methods typically track by performing an iterative search for the local maxima of a similarity cost function between the template image and the current image. The cost function widely used is the sum of squared differences (SSD) between the template and the current image such as in [85, 62, 3]. More robust similarity measures have been applied and the mean-shift algorithm or other optimization techniques have been utilized to find the optimal solution [11, 15, 24, 37, 19]. In Chapter 4, we presented a new similarity measure in the joint feature-spatial spaces, where the mean-shift algorithm is used for object tracking.

On the other hand, the stochastic methods use the state space to model the

underlying dynamics of the tracking system. In a linear-Gaussian model with linear measurement, there is always only one mode in the posterior probability density function (p.d.f.), the Kalman filter propagates and updates the mean and covariance of the distribution. For nonlinear or non-Gaussian problems, it is impossible to evaluate the distributions analytically and many algorithms have been proposed to approximate them. The particle filter, also known as sequential Monte Carlo [32], is the most popular approach which recursively constructs the posterior p.d.f. of the state space using Monte Carlo integration. It has been developed in the computer vision community and applied to tracking problem and is also known as the Condensation algorithm [71].

The particle filter based tracking algorithms usually use contours, color features, or appearance models [71, 72, 128, 77]. The color histogram is robust against noise and partial occlusion, but suffers from illumination changes, or the presence of the confusing colors in the background. Most of all, it ignores the spatial layout information. The computation is expensive if the tracked region and the number of samples are large. The contour-based methods are invariant against the illumination variation but computationally expensive which restricts the number of samples (particles). Unfortunately when the dimensionality of the state space increases, the number of samples required for the sampling increases exponentially.

To resolve these problems, we adopt the Harr-like rectangle features introduced by Viola and Jones [120] for object detection, and the edge orientation histogram. The rectangle features can be efficiently evaluated by a few table lookup operations from the integral image as shown in [120]. The Harr-like rectangle features yield

satisfactory results unless there are confusing pieces of the background with color similar to the foreground, or if there are significant illumination changes.

The edge or contour features are more robust to illumination variation or presence of confusing background colors, but are sensitive to clutter. One natural way to improve both the color and edge features is to combine them as in [11, 72, 128]. However, these earlier methods dealing with the edges are either time-consuming or restricted to simple shape models. To provide a general way to treat edge features instead we manipulate them using the edge orientation histogram (EOH) [47]. There are two reasons for this: one is that they can be efficiently computed in the same way as the rectangle features using the integral image; the other is that they are robust to scene illumination changes and provides more information than a simple contour model. Since both features can be evaluated efficiently, we can generate many more samples to alleviate the curse of the dimensionality and improve robustness.

While evaluating the probability that samples in the target image appear from the distribution of the original object, we will find that for most samples the probability is extremely low, and they can be pruned immediately by a bootstrap step in the particle filter. The few surviving samples are subjected to a more careful scrutiny in the following stages. In these following stages, more complicated and more discriminative features can be used to remove ambiguities in the first stage. This coarse-to-fine cascade idea, where the first stages reject most incorrect matches, while retaining all correct and a few incorrect ones, has been successfully used in target recognition and face detection [136, 45, 81, 120, 103]. As will be seen it allows for significant speed-up.

While employing extensively the integral image and the coarse-to-fine cascade scheme, our method is totally different from the object detection algorithms or face detection such as [136, 45, 81, 120, 103]. Temporal correlation in the sequences is ignored in these detection algorithms. If they are naively applied to object tracking, the false alarms or false negatives are too high for continuous tracking applications.

A further speed-up is made possible by improving the convergence of the Monte Carlo integration in the particle filter. We do so by using a quasi-random generator to generate the sample points [93]. To improve the running of the tracking algorithm we also employ the Streaming SIMD Extensions (SSE) and SSE2 instructions in the Pentium 4 processor. These are especially useful to improve the performance of computing the integral images. All these techniques allow our algorithm to comfortably run in real-time on a PC desktop.

6.2 Particle Filter

The particle filter is a Bayesian sequential importance sampling technique, which recursively approximates the posterior distribution using a finite set of weighted samples. It consists of essentially two steps: prediction and update. Given all available observations $\mathbf{z}_{1:t-1} = \{\mathbf{z}_1, \ldots, \mathbf{z}_{t-1}\}$ up to time $t-1$, the prediction stage uses the probabilistic system transition model $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ to predict the posterior at time t as

$$p(\mathbf{x}_t|\mathbf{z}_{1:t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|\mathbf{z}_{1:t-1})d\mathbf{x}_{t-1}. \tag{6.1}$$

At time t, the observation $\mathbf{z}_t$ is available, the state can be updated using Bayes' rule

$$p(\mathbf{x}_t|\mathbf{z}_{1:t}) = \frac{p(\mathbf{z}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{z}_{1:t-1})}{p(\mathbf{z}_t|\mathbf{z}_{1:t-1})}, \tag{6.2}$$

where $p(\mathbf{z}_t|\mathbf{x}_t)$ is described by the observation equation.

In the particle filter, the posterior $p(\mathbf{x}_t|\mathbf{z}_{1:t})$ is approximated by a finite set of N samples $\{\mathbf{x}_t^i\}_{i=1,\ldots,N}$ with importance weights w_t^i. The candidate samples $\tilde{\mathbf{x}}_t^i$ are drawn from an importance distribution $q(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t})$ and the weight of the samples are

$$w_t^i = w_{t-1}^i \frac{p(\mathbf{z}_t|\tilde{\mathbf{x}}_t^i)p(\tilde{\mathbf{x}}_t^i|\mathbf{x}_{t-1}^i)}{q(\tilde{\mathbf{x}}_t|\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t})}. \tag{6.3}$$

The samples are resampled to generate an unweighted particle set according to their importance weights to avoid degeneracy. In the case of the bootstrap filter [32], $q(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \mathbf{z}_{1:t}) = p(\mathbf{x}_t|\mathbf{x}_{t-1})$ and the weights become the observation likelihood $p(\mathbf{z}_t|\mathbf{x}_t)$.

6.3 Observation Models

The observation model is is used to measure the observation likelihood of the samples, and is an important issue for object tracking. Many observation models have been built for particle filtering tracking. In [71], a contour based appearance template is chosen to model the target. The tracker based on a contour template gives an accurate description of the targets but performs poorly in clutter and is generally time-consuming. The initialization of the system is relatively difficult and tedious. In contrast, color-based trackers are faster and more robust, where the color histogram is typically used to model the targets to combat the partial occlusion, and

non-rigidity [92, 128, 11, 24]. The drawback of the color histogram is that spatial layout is ignored, and the trackers based on it are easily confused by a background with similar colors.

The combination of the two features provides better performance at the price of trading speed for robustness. The color features provide robust and discriminative description of the objects using the body information. The edge features provide a discriminative description of the objects using the boundary information. Such a combination significantly improves the robustness and discriminative power of the features at the price of high computational load and slow tracking speed.

In order to resolve the contradiction between the robustness and the tracking speed, we use the simple rectangle features which have been used by Viola and Jones in object detection [120] and can be used in the bootstrap step as discussed below. These simple features can be efficiently evaluated by several table lookup operations on the integral image. The features are obtained on the grayscale images and can be easily extended to color images.

The edge information is represented using the edge orientation histogram (EOH). The EOH has been widely used for gesture recognition [47], pose estimation [105], distinctive image feature extraction [84], and object detection [80]. The reasons for using the EOH is that it is invariant to scene illumination changes, is discriminative against a background with confusing colors, and is simple and fast to compute.

6.3.1 Color Rectangle Features

The rectangle features were introduced by Viola and Jones for real-time object detection [120]. In their method, the grayscale image was converted to integral image format (an image in which at each pixel the value is the sum of all pixels above and to the left of the current position). The sum of the pixels within any rectangle can then be computed in four table lookup operations on the integral image. The color images can be treated as multi-channel intensity images to generate multi-channel integral images. The computation can be speeded up by using the SSE/SSE2 instructions that are available on the Pentium 4 CPU, where 128-bit instructions can be used to manipulate four 32-bit integers simultaneously.

To model the target using color information, we pick n rectangular regions $R_1, \ldots, R_n$ within the object to be tracked. Each rectangle R_i is represented by the mean (r, g, b) color of the pixels within region R_i (other color spaces can be considered similarly)

$$(r_i, g_i, b_i) = \sum_{(x,y)\in R_i} \left(r(x,y), g(x,y), b(x,y)\right) / A_i, \tag{6.4}$$

where A_i is the number of pixels within R_i. The mean color vector $(r_i^\star, g_i^\star, b_i^\star)$ of each region R_i can be computed during initialization. The reason we choose this color representation instead of the popular color histogram is that it encodes the spatial layout of the targets and offers robustness against noise. Furthermore, most of the targets consist of several homogenous sub-regions which makes the color histogram an inefficient representation.

Such a color representation of the targets has been used in [43] for head track-

ing, where a hypothesize-and-test procedure is used to find a match between frames. In [43], for real time performance they can only consider a relatively small number of hypotheses, since there was no efficient way to evaluate the rectangle features. This leads to decreased robustness. Here we have such an efficient way and can consider more hypotheses.

If we denote $\mathbf{k}^\star = \{(r_i^\star, g_i^\star, b_i^\star)\}_{i=1,\ldots,n}$ as the reference color model and $\mathbf{k}(\mathbf{x}_t)$ as the candidate color model, the similarity between $\mathbf{k}^\star$ and $\mathbf{k}(\mathbf{x}_t)$ can be measured by the Euclidean distance between them

$$\rho(\mathbf{k}^\star, \mathbf{k}(\mathbf{x}_t)) = \left[\sum_{i=1}^{n} (r_i^\star - r_i)^2 + (g_i^\star - g_i)^2 + (b_i^\star - b_i)^2\right]^{\frac{1}{2}}. \tag{6.5}$$

The likelihood distribution is given by

$$p(\mathbf{z}_t|\mathbf{x}_t) \propto e^{-\rho^2(\mathbf{k}^\star, \mathbf{k}(\mathbf{x}_t))/\sigma^2}, \tag{6.6}$$

where $\sigma = 10$ in our experiments.

The number of the rectangles within the object can be as small as two in the first stage which we will discuss later in this chapter. There are two reasons for such a setup: one is efficiency, the other is that we want to keep more candidates in the first stage for robustness and prune those majority negative samples as soon as possible. This strategy has been proven successful in object detection [120], and we observe the same for tracking.

6.3.2 Edge Orientation Histogram

The color features are reliable for most tracking tasks, even when there is occlusion or overlap. However, it may perform poorly if the background presents

confusing colors. Many other feature types have been proposed for combination with color. Isard and Blake [72] have shown that the combination of color with a contour model gives faster and more robust tracking. The color histogram and an ellipse shape model are combined for object tracking [11, 128]. In this chapter, we use the edge orientation histogram for the purpose of simplicity, efficiency and generalization. It has been widely used in a variety of vision applications [47, 105, 84, 80, 26].

To detect edges, we first convert color images to grayscale intensity images. Edges are detected using the horizontal and vertical Sobel operators: K_x and K_y [46]:

$$G_x(x,y) = K_x * I(x,y), \quad G_y(x,y) = K_y * I(x,y). \tag{6.7}$$

The strength and the orientation of the edges are

$$S(x,y) = \sqrt{G_x^2(x,y) + G_y^2(x,y)}, \tag{6.8}$$

$$\theta = \arctan\left(G_y(x,y)/G_x(x,y)\right). \tag{6.9}$$

We also apply a threshold T to $S(x,y)$ to remove noise (T is set to 100 in our experiments). The edges are counted into K bins with their strengths $S(x,y)$. Figure 6.1 shows an example of the global edge orientation histogram of the walker in the image.

The edge orientation histogram can be built without explicitly computing the angles of the edges. Instead we use the normalized horizontal and vertical strengths $g_x = G_x/S$ and $g_y = G_y/S$ to index the edges into K bins. The algorithm gives

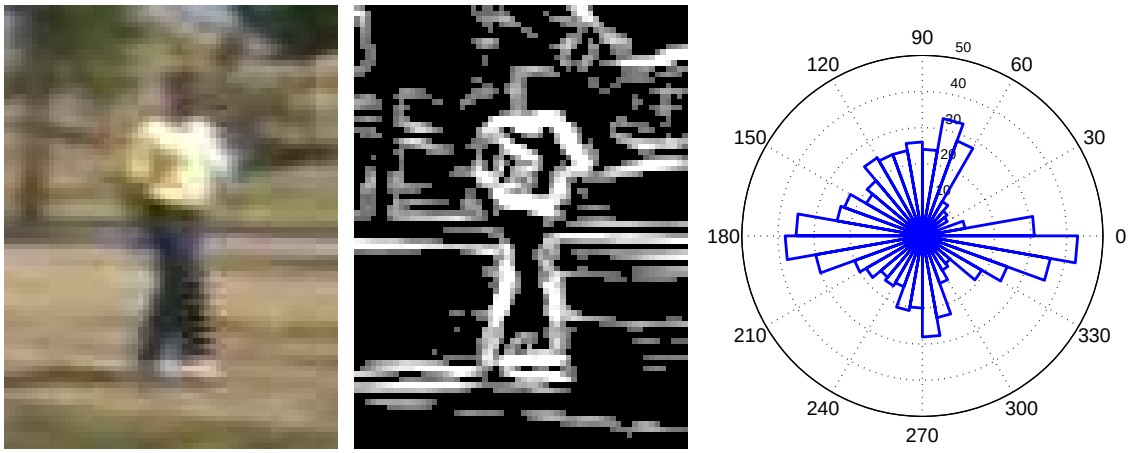


Figure 6.1: Edge orientation histogram. (*Left*) Example image. (*Center*) Edge strength image. (*Right*) Polar plot of edge orientation histogram.

satisfactory results even when K is as small as 4.

The edge orientation histogram within a rectangle region can be efficiently computed by treating it as K separated channels and accumulating K integral images. The i-th bin value within a rectangle is the sum computed by four table lookup operations on i-th integral image.

The similarity between the template and the current image is computed with the Euclidean distance between the two global edge orientation histograms as in [84]. The observation likelihood is calculated similarly using an expression such as (6.6).

6.3.3 Cascade of Features

The combination of the color information and edge orientation histogram achieves excellent performance in term of speed and accuracy. Typically the scores of about 90 of samples are almost zero (see the example in figure 6.2). The remaining

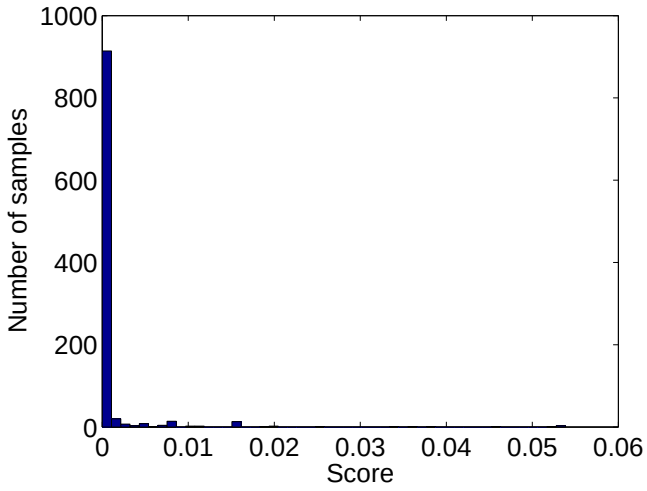


Figure 6.2: The histogram of scores of 1000 samples from an example sequence. Most samples have a vanishingly small score.

samples are subjected to a more careful examination. This cascade-like method has been widely used in many applications such as [136, 45, 81, 120, 103]. In particular, Viola and Jones [120] construct a cascade of classifiers which yields a very fast face detector. The use of cascade is effective because a majority of the sub-regions are negative. The cascade tries to reject as many as possible at the earlier stage, to concentrate the algorithm effort on the positive sub-regions.

Since the probability of most samples is almost zero, we can cut them off from the second stage evaluation. More sophisticated and more discriminative features and observation models are used in the second stage. In our implementation, we use the *two-rectangle*, *three-rectangle* and *four-rectangle* features in [120] as shown in Figure 6.3. Those features were used by the classifiers in the face detector [120]. We use them in the second stage because they are more discriminative at the same time more sensitive to the presence of edges, change in illumination or noise. The

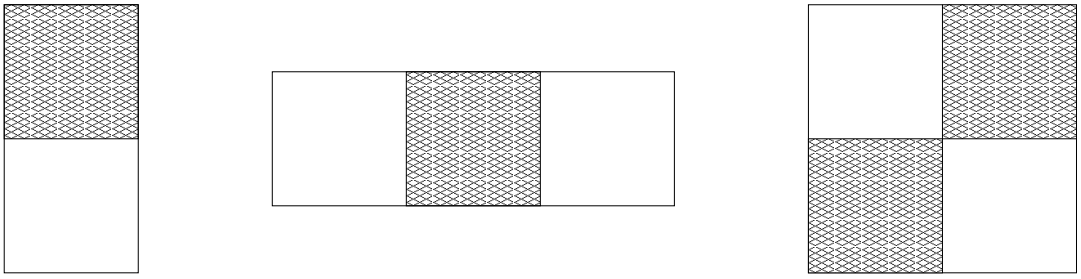


Figure 6.3: Rectangle features used here, which are the remainders of the sums of the white rectangles from the gray ones. (*Left*) two-rectangle feature. (*Center*) three-rectangle feature. (*Right*) four-rectangle feature.

likelihood distribution of the subset of samples can be calculated similarly as in the first stage, then multiplied by the likelihood of the first stage. The probability of the remaining samples is scaled down by the smallest likelihood from the second stage. The samples with significant low probability will be neglected in the following stages.

The features in the third stage is the edge orientation histogram which is similar to the one in the Scale Invariant Feature Transform (SIFT) descriptor [84]. The SIFT descriptor has achieved great success as a method for reliable image matching. The region is divided into $m \times n$ subregions and local histograms with 4 orientation bins are constructed in each by using the orientation integral images. The local edge orientation histograms are stacked into a multidimensional feature vector. The similarity between the template and the current image is computed using the Euclidean distance between the two multidimensional feature vectors. The observation likelihood is calculated using the expression (6.6). Each likelihood is multiplied by

the previous one and the others are multiplied by the smallest likelihood in the third stage. In principle, more complicated features or representations can be used to construct the further levels of the cascade, but we restrict ourselves to these here.

6.4 Particle Filter Tracking

The proposed particle filter tracker consists of an initialization of the template model and a sequential Monte Carlo implementation of a Bayesian filtering for the stochastic tracking system. In each iteration, the particle filter tracking algorithm consists of two steps: prediction and update.

The state of the particle filter is defined as $\mathbf{x} = (x, y, s_x, s_y)$, where x, y indicate the location of the target, s_x, s_y the scales in the x and y directions. In the prediction stage, the samples in the state space are propagated through a dynamic model. The dynamics usually is an auto-regressive process (AR). We use a first-order AR model for fair comparison and simplicity:

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{v}_{t-1}, \tag{6.10}$$

where $\mathbf{v}_{t-1}$ is a multivariate Gaussian random variable.

To draw samples from the normal distribution, we use the quasi-random sequence generator which converges in rate of $(\ln N)^d/N$ in d-dimensional state space instead of $N^{-1/2}$ using the pseudo-random sequence generator [93, 68]. Figure 6.4 shows the random dots generated by the two random sequence generators. Clearly, the quasi-random sequence is more symmetric and samples space with less discrepancy.

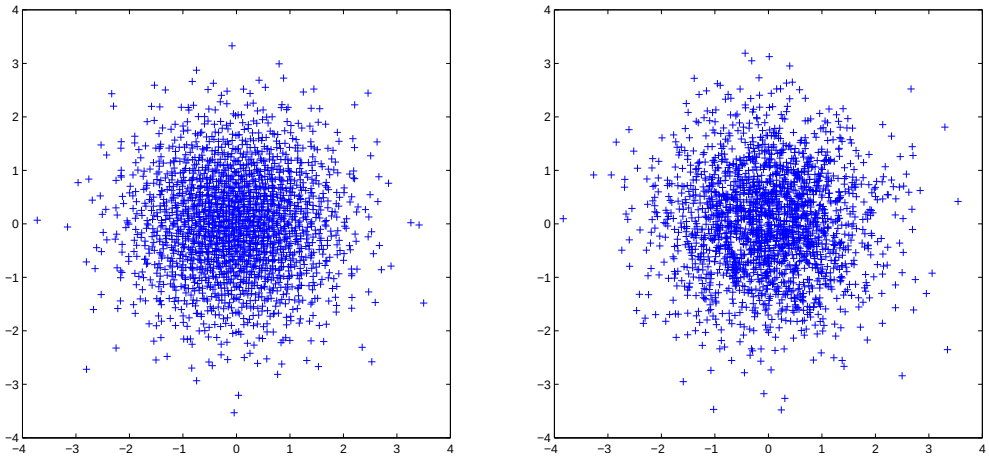


Figure 6.4: 2048 points of two-dimensional Gaussian random sequences generated by (*Left*) quasi-random sequence generator and (*Right*) pseudo-random sequence generator. The quasi-random sequence is more symmetric and has less discrepancy.

The update stage applies the observation models to estimate the observation likelihood for each samples, i.e., the weights of samples in the case of the bootstrap filter. Since the probability of most samples is negligible, a bootstrap resampling is necessary to avoid the degeneracy. We apply the bootstrap scheme in [94] which uses order statistics and has the computational complexity $O(N)$.

6.5 Experiments

The proposed particle filter based tracker has been implemented in C and tested on a 1.4GHz Pentium4 PC with 512MB memory. It has been applied to a variety of tracking scenarios and tasks, including single and multiple object tracking.

6.5.1 Single Object Tracking

In the first experiment, a single person in a video sequence is walking in an office environment under severe illumination condition. The image size of the sequence is 352×240. The top row in Figure 6.5 shows that the tracker follows the target consistently and robustly despite the large illumination variations. The average tracking time per frame is about $0.015s$ with 1000 samples. The bottom row in Figure 6.5 shows the tracking results of the color histogram based tracker in [92] with the same system dynamics, same initial template and 100 samples. The tracking rate is about $50fps$. The tracker totally loses the target after several frames. With 1000 samples, the tracking speed for the original algorithm will be very low (about $5fps$).

In the second experiment, the sequence is captured from outdoor environment with cluttered background and large changes in body size and shape. The results are shown in Figure 6.6. The image size is 720×480 and the number of particles is 1000. The average tracking rate is about $30fps$. Here we compute the integral image from the whole original image. If we use part of the image, the tracking speed will be faster. The great benefit of the proposed algorithm over other particle filter based tracking methods is shown in the multiple object tracking.

6.5.2 Multiple Object Tracking

In the third experiment, we use the proposed algorithm to simultaneously track multiple objects. Each object is associated with a individual template and 1000

Figure 6.5: Particle filter based tracking in an office environment under severe illumination condition. (*Top*) The results of the proposed tracker. (*Bottom*) The results of the color histogram based tracker.

Figure 6.6: The results of the proposed particle filter based tracking for an outdoor sequence.

particles. Each object moves independently. The proposed algorithm successfully tracked all objects through all frames. The image size is 352×288. Examples of the tracking results are shown in Figure 6.7. The tracking time with respect to the frame index is shown in the left panel of Figure 6.8. We also measured the average tracking time for single object with respect to the number of particles which is shown in the right panel of Figure 6.8. The same procedures and configurations are applied to the color histogram based tracker and the corresponding tracking time is shown in Figure 6.8. It indicates that the increase of the time for the proposed algorithm with respect to the number of particles is much slower than for the color histogram based method. Most of the time in our method is spent on building the integral images. Once the integral images are built, the evaluation of the observation likelihood by the proposed method is independent of the size of regions and is very efficient. In contrast, for the color histogram based method or other methods, the bottle-neck is the building of the histograms whose complexity is proportional to the number of particles and the size of the regions.

6.6 Conclusions

In this chapter, we proposed an efficient and robust particle filter based object tracking algorithm. The particle filter maintains multiple hypotheses about the state of the tracked objects by representing the state space by a set of weighted samples. In general, the more samples and richer target representation, the better chances the tracking algorithms succeed in cluttered and noisy environments. However,

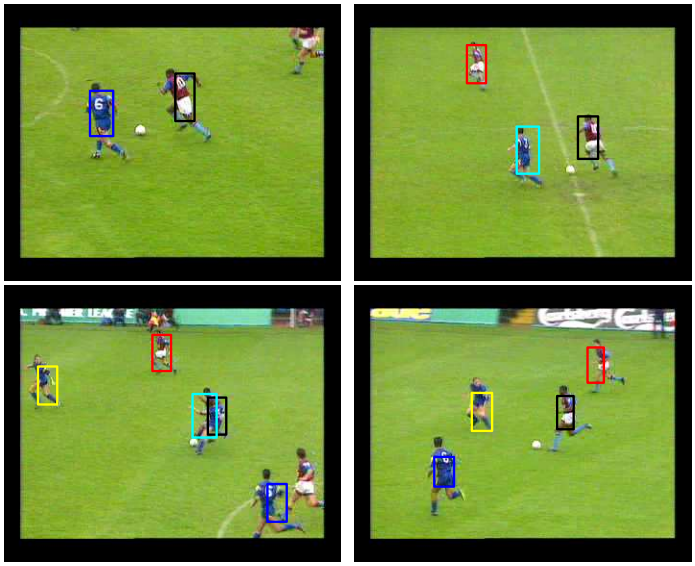


Figure 6.7: Results of the proposed particle filter based multiple object tracking for the football game sequence.

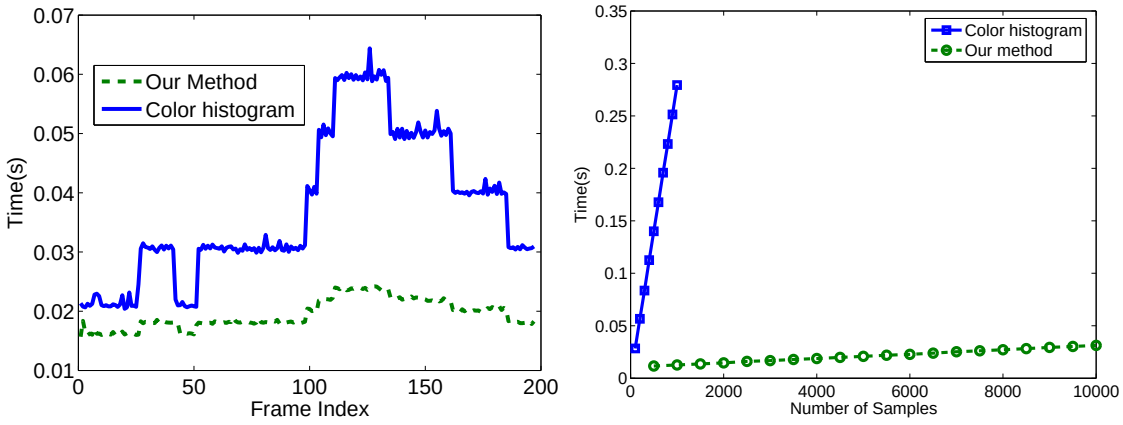


Figure 6.8: (*Left*) Tracking time with respect to the frame index. (*Right*) Tracking time with respect to the number of samples.

they can be very inefficient when the popular color histograms, shapes, contours or a combination of these are used to evaluate the observation likelihood. We use the rectangle features and edge orientation histogram to evaluate the observation likelihood. They can be efficiently computed with integral images. To increase the discriminative capacity while speeding up the evaluation, we adopt a cascaded scheme which results in highly discriminative observation likelihood. The Monte Carlo integration in the particle filter converges at rate $O(N^{-1/2})$. We use the quasi-random sampling to improve the convergence rate to $O((\log N)^d/N)$ as in [93].

The above improvements make the tracking algorithm very efficient and robust against clutter, illumination changes and short period time occlusions. From the experiments, we find most of computations are spent on building the integral images. But once they are built, the evaluation of the observation likelihood can be done using several table lookup operations. So we can generate much more samples to represent the state space more accurately, which make it very suitable for multiple object tracking as in [89]. The probabilistic exclusion principle proposed by MacCormick and Blake [86] is very useful for tracking multiple objects, which employs the partition sampling to reduce the high computational cost from fully coupled systems. A combination of MacCormick and Blake's technique and ours will make the multiple object tracking more efficient and more robust. The object detection algorithm [120] also can be naturally integrated with our method. Currently the orientation of the targets is fixed during the tracking. The possible solution is to assign an orientation to the tracked objects using the edge orientation histograms as in [84].

Chapter 7

Conclusions and Future Work

The kernel methods have succeeded in many research areas, including vision and learning, where the key computation is the kernel density estimate. Unfortunately, a drawback is that the direct computation of a kernel density estimate of N data points requires $O(N^2)$ operations. The fast Gauss transform can be used to reduce the cost to linear order in low-dimensional spaces. In higher dimensions, we proposed the improved fast Gauss transform to speed up the computation of the kernel density estimates. The IFGT has been applied to object tracking and machine learning applications.

7.1 Future Work

In this section we outline some possible directions for future work. We start with the improved fast Gauss transform, then object tracking.

7.1.1 Improved Fast Gauss Transform

Currently the improved fast Gauss transform uses no hierarchical data structures such as trees. However for large amounts of data, such tree data structure is very useful. We can first subdivide the space into small cells using k-center algorithm, then build a tree upon these cells [88]. Such scheme will reduce the cost of

indexing at the same time providing more fine resolution to the data.

Yet another possible improvement to the algorithm is to consider the nearest neighbor searching. For nearest neighbor searching, there are two types of important techniques: tree-based methods [9, 27] and locality-sensitive hashing [67]. The combination of the data structures in the nearest neighbor searching with the improved fast Gauss transform would be a interesting topic in the future work.

Bandwidth choice is very important to the performance of the kernel density estimate. A good bandwidth is not easy to obtain. Usually it takes a large amount of computation by the way of cross-validation. The improved fast Gauss transform will accelerate this step substantially. In many cases the variable bandwidth is beneficial to the kernel density estimates. Any progress on this topic will be very interesting [100].

7.1.2 Object Tracking

For object tracking, as pointed in [62], the difficulty comes from three principle sources: target deformations, illumination changing, and occlusions of the target. For target deformation and occlusion, the tracking algorithm in Chapter 4 works well. To further improve the robustness of the algorithm, we should consider the illumination changes and the template adaption. Such adaption should be considered in the probabilistic sense. The methods proposed in [6, 62] can be applied to the current tracking algorithm.

One major problem in tracking is the interference from the background. When

the background and the target share the same colors or intensity, it is very difficult for the tracking algorithm to work properly. A good background model will greatly benefit the tracking algorithm and help to detect the objects [41]. We will consider probabilistic reasoning methods to integrate the background information into the tracking algorithm.

For multiple object tracking, the interaction among the objects is very helpful for tracking. Recently the collaborative tracking [137] gives a way for visual tracking of multiple targets. To completely solve this problem, we should pay more attention to the distinctive features [84] and multiple-cue techniques. Currently we used fixed templates for object tracking. Usually an adaptive template will improve the performance of the trackers. But there is always a risk that the adaptive template will drift away from the target. The solution to this drifting problem is currently an active research topic [75].

7.1.3 Kernel Based Contour Tracking

The problem of representing shapes or contours is a challenging task in image processing and computer vision. The active contour model (or snake) proposed by Kass *et al.* [76] has been influential in many applications, including image segmentation, motion tracking, and stereo matching. The basic idea of active contours is to dynamically evolve a spline curve driven by a combination of an internal spline energy term, external constraint force terms, and image forces that attract the contour toward desired features such as edges or lines. The evolution of the curve is

realized by an energy minimization procedure. Despite the success it achieved, there are several deficiencies in the method. First it is numerically unstable, because there is a need for estimates of high-order derivatives. Second, the contours easily drift toward strong outlier edges. Third, the snake is not easily extended to dimensions higher than two. Lastly, it does not handle topological changes in the shapes well.

Many methods have been proposed to fix these problems. Amini *et al.* [1] proposed an algorithm for minimizing the energy of active contours using dynamic programming. Their method is numerically more stable, and allows for hard constraints to be enforced on the active contours to achieve desirable behavior. Later Williams and Shah [125] speeded up the algorithm from order $O(nm^3)$ to order $O(nm)$, where n is the number of the contour points and m the size of searching neighborhood.

To accommodate the characteristics or structures of objects, learning methods have been used for deforming the model in ways consistent with the training samples. Cootes *et al.* [25] proposed Active Shape Models (ASMs) which learn patterns of variability of the objects from a set of training samples. The models are deformed in ways describing the class of represented objects. The method iteratively searches for the better fit to the data using a Point Distribution Model (PDM), which is similar to Lowe's refinement technique [83]. Compared with Active Contour Models (Snakes), it is more robust against noise, clutter and occlusion. However, it is a tedious pre-processing task to create the models and find the landmark points in the images.

The active contour models and active shape models use the landmark points

to represent the boundary of the objects. These point-based approaches have difficulty to describe the topological changes in the models, no direct way to perform the inside-outside test key in applications, and difficulty representing shapes in dimensions higher than two. The inside-outside test is a query that determines whether the query point lies inside or outside a contour or surface. It is important for determining the intersection or evaluating the objective function value for tracking. All these are the reasons why the level set methods [90] have gained popularity recently. It has been shown that level set methods are numerical more stable, can handle sharp corners and topological changes [16].

However, level set methods are too computationally expensive and conceptually involved for real-time object tracking. A possible solution is to represent the shapes in a high-dimensional Reproducing Kernel Hilbert Space (RKHS) [121] using the kernel method. Such a new shape representation will be stable, flexible, easily extended in to higher dimensions and easy to perform inside-outside test. It is also easily integrated in a particle filter framework for dynamic deformable shape tracking. Prior information on the shapes can be learnt using the embedded kernel machines and the smoothness of the shapes can be controlled by a regularization term.

BIBLIOGRAPHY

[1] Amir A. Amini, Terry E. Weymouth, and Ramesh C. Jain. Using dynamic programming for solving variational problems in vision. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(9):855–867, September 1990.

[2] Shai Avidan. Support vector tracking. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, volume I, pages 184–191, Kauai, HI, 2001.

[3] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *Int'l Journal of Computer Vision*, 56(3):221–255, February 2004.

[4] Dana H. Ballard and Christopher M. Brown. *Computer Vision*. Prentice-Hall, Englewood Cliffs, NJ, 1982.

[5] Benedicte Bascle and Rachid Deriche. Region tracking through image sequences. In *Proc. Int'l Conf. Computer Vision*, pages 302–307, 1995.

[6] Ronen Basri and David Jacobs. Lambertian reflectance and linear subspaces. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(2):218–233, February 2003.

[7] B. J. C. Baxter and G. Roussos. A new error estimate of the fast Gauss transform. *SIAM Journal on Scientific Computing*, 24(1):257–259, 2002.

[8] Richard Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[9] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[10] Marshall Bern and David Eppstein. Approximation algorithms for geometric problems. In D. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 8, pages 296–345. PWS Publishing Company, Boston, MA, 1997.

[11] Stan Birchfield. Elliptical head tracking using intensity gradients and color histograms. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, pages 232–237, Santa Barbara, CA, 1998.

[12] Michael Black, Guillermo Sapiro, David Marimont, and David Heeger. Robust anisotropic diffusion. *IEEE Trans. Image Processing*, 7(3):421–432, March 1998.

[13] Michael J. Black and Allan D. Jepson. Eigentracking: Robust matching and tracking of articulated objects using a view-based representation. *Int'l Journal of Computer Vision*, 26(1):63–84, 1998.

[14] John Board and Klaus Schulten. The fast multipole algorithm. *Computing in Science & Engineering*, 2(1):76–79, 2000.

[15] Gary Bradski. Computer vision face tracking for use in a perceptual user interface. *Intel Technology Journal*, 2(Q2), 1998.

[16] Tony F. Chan and Luminita A. Vese. Active contours without edges. *IEEE Trans. Image Processing*, 10(2):266–277, February 2001.

[17] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(8):790–799, 1995.

[18] German Cheung, Simon Baker, and Takeo Kanade. Shape-from-silhouette of articulated objects and its use for human body kinematics estimation and motion capture. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, volume I, pages 77–84, Madison, WI, 2003.

[19] Robert Collins. Mean-shift blob tracking through scale space. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, volume II, pages 234–240, 2003.

[20] Dorin Comaniciu. Bayesian kernel tracking. In *Annual Conf. of the German Society for Pattern Recognition*, pages 438–445, Zurich, Switzerland, 2002.

[21] Dorin Comaniciu and Peter Meer. Mean shift analysis and applications. In *Proc. Int'l Conf. Computer Vision*, pages 1197–1203, 1999.

[22] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603 – 619, May 2002.

[23] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Real-time tracking of non-rigid objects using mean shift. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, volume II, pages 142–149, 2000.

[24] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Kernel-based object tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(5):564–577, May 2003.

[25] Timothy F. Cootes, Christopher J. Taylor, D. H. Cooper, and J. Graham. Active shape models – their training and application. 61(1):38–59, January 1995.

[26] James W. Davis. Hierarchical motion history images for recognizing human motion. In *IEEE Workshop on Detection and Recognition of Events in Video*, pages 39–46, Vancouver, Canada, July 2001.

[27] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 2nd edition, 2000.

[28] Pierre Devijver and Josef Kittler. *Pattern Recognition: A Statistical Approach*. Prentice-Hall International, London, 1982.

[29] L. Devroye and F. Machell. Data structures in kernel density estimation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 7(3):360–366, 1985.

[30] Jack Dongarra and Francis Sullivan. The top 10 algorithms. *Computing in Science & Engineering*, 2(1):22–23, 2000.

[31] David L. Donoho. High-dimensional data analysis: The curses and blessings of dimensionality. AMS Conference: Math Challenges of the 21st Century, August 2000.

[32] Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, New York, 2001.

[33] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley & Sons, New York, 2nd edition, 2001.

[34] Ramani Duraiswami, Nail A. Gumerov, Dmitry N. Zotkin, and Larry S. Davis. Efficient evaluation of reverberant sound fields. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 203–206, New Palz, NY, 2001.

[35] Frédo Durand and Julie Dorsey. Fast bilateral filtering for the display of high-dynamic-range images. In *ACM SIGGRAPH*, pages 257–266, San Antonio, TX, 2002.

[36] A. Elgammal, R. Duraiswami, and L. Davis. Efficient non-parametric adaptive color modeling using fast Gauss transform. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, Kauai, Hawaii, 2001.

[37] Ahmed Elgammal, Ramani Duraiswami, and Larry Davis. Probabilistic tracking in joint feature-spatial spaces. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, volume I, pages 781–788, Madison, WI, 2003.

[38] Ahmed Elgammal, Ramani Duraiswami, and Larry S. Davis. Efficient computation of kernel density estimation using fast Gauss transform with applications for segmentation and tracking. In *Int. workshop on Statistical and Computational Theories of Vision*, Vancouver, Canada, July 2001.

[39] Ahmed Elgammal, Ramani Duraiswami, and Larry S. Davis. Efficient non-parametric adaptive color modeling using fast Gauss transform. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, pages 563–570, Kauai, Hawaii, 2001.

[40] Ahmed Elgammal, Ramani Duraiswami, and Larry S. Davis. Efficient kernel density estimation using the fast Gauss transform with applications to color modeling and tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(11):1499–1504, November 2003.

[41] Ahmed Elgammal, Ramani Duraiswami, David Harwood, and Larry Davis. Background and foreground modeling using nonparametric kernel density estimation for visual surveillance. *Proceedings of the IEEE*, 90(7):1151–1163, July 2002.

[42] Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proc. 20th ACM Symp. Theory of computing*, pages 434–444, Chicago, Illinois, 1988.

[43] P. Fieguth and D. Terzopoulos. Color based tracking of heads and other mobile objects at video frame rates. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, pages 21–27, Puerto Rico, 1997.

[44] Shai Fine and Katya Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, December 2001.

[45] Francois Fleuret and Donald Geman. Coarse-to-fine face detection. *Int'l Journal of Computer Vision*, 41(1/2):85–107, 2001.

[46] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach.* Prentice Hall, 2003.

[47] William T. Freeman and Michal Roth. Orientation histograms for hand gesture recognition. In *Proc. Int'l Workshop on Automatic Face and Gesture Recognition*, pages 296–301, Zurich, Switzerland, June 1995.

[48] K. Fukunaga and R. R. Hayes. The reduced Parzen classifier. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(4):423–425, 1989.

[49] K. Fukunaga and L. D. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Trans. Inform. Theory*, 21(1):32–40, 1975.

[50] Glenn Fung and Olvi L. Mangasarian. Proximal support vector machine classifiers. In *Proceedings KDD-2001: Knowledge Discovery and Data Mining*, pages 77–86, San Francisco, CA, 2001.

[51] Federico Girosi, Michael Jones, and Tomaso Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7(2):219–269, 1995.

[52] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 2002.

[53] Teofilo Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[54] Leslie Greengard. *The rapid evaluation of potential fields in particle systems*. MIT Press, Cambridge, MA, 1988.

[55] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73(2):325–348, 1987.

[56] Leslie Greengard and John Strain. A fast algorithm for the evaluation of heat potentials. *Comm. Pure Appl. Math.*, 43(8):949–963, 1990.

[57] Leslie Greengard and John Strain. The fast Gauss transform. *SIAM J. Sci. Statist. Comput.*, 12(1):79–94, 1991.

[58] Nail A. Gumerov and Ramani Duraiswami. Fast, exact, and stable computation of multipole translation and rotation coefficients for the 3-d Helmholtz equation. Technical Report CS-TR-4264, UMIACS, University of Maryland, College Park, MD, 2001.

[59] Nail A. Gumerov and Ramani Duraiswami. Computation of scattering from n spheres using multipole reexpansion. *J. Acoust. Soc. Am.*, 112(6):2688–2701, 2002.

[60] Nail A. Gumerov, Ramani Duraiswami, E. A. Borovikov, and Larry S. Davis. Data structures, optimal choice of parameters, and complexity results for generalized fast multipole methods in d dimensions. working paper to be submitted to J. Comput. Phys., 2002.

[61] Nail A. Gumerov, Ramani Duraiswami, and Eugene A. Borovikov. Data structures, optimal choice of parameters, and complexity results for generalized multilevel fast multipole methods in d dimensions. Technical Report UMIACS-TR-2003-28, UMIACS, University of Maryland, College Park, 2003.

[62] Gregory Hager and Peter Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(10):1025–1039, 1998.

[63] Richard I. Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Cambridge, UK, 2nd edition, 2004.

[64] D. S. Hochbaum and D. B. Shmoys. A best possible heuristic for the k-center problem. *Mathematics of Operations Research*, 10(2):180–184, 1985.

[65] Berthold Klaus Paul Horn. *Robot Vision*. The MIT Press, Cambridge, MA, 1986.

[66] Peter Huber. *Robust Statistics*. John Wiley & Sons, 1981.

[67] Piotr Indyk. Nearest neighbors in high-dimensional spaces. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 39. CRC Press, 2rd edition, 2004.

[68] Intel. *Vector Statistical Library Notes*, 5.0 edition, April 2004.

[69] Michal Irani and Shmuel Peleg. Motion analysis for image enhancement: Resolution, occlusion, and transparency. *JVCIP*, 4:324–335, Dec. 1993.

[70] Michael Isard and Andrew Blake. Contour tracking by stochastic propagation of conditional density. In *Proc. European Conf. Computer Vision*, pages 343–356, Cambridge, UK, 1996.

[71] Michael Isard and Andrew Blake. Condensation – conditional density propagation for visual tracking. *Int'l Journal of Computer Vision*, 29(1):5–28, August 1998.

[72] Michael Isard and Andrew Blake. Icondensation: Unifying low-level and high-level tracking in a stochastic framework. In *Proc. European Conf. Computer Vision*, volume 1, pages 893–908, 1998.

[73] David Jacobs, Daphna Weinshall, and Yoram Gdalyahu. Class representation and image retrieval with non-metric distances. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(6):583–600, 2000.

[74] B. Jeon and D. A. Landgrebe. Fast Parzen density estimation using clustering-based branch and bound. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(9):950–954, 1994.

[75] Allan D. Jepson, David J. Fleet, and Thomas F. El-Maraghi. Robust online appearance models for visual tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(10):1296–1311, October 2003.

[76] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *Int'l Journal of Computer Vision*, 1:321–331, 1988.

[77] Zia Khan, Tucker Balch, and Frank Dellaert. A Rao-Blackwellized particle filter for eigentracking. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, volume 2, pages 980–986, Washington DC, 2004.

[78] Michel LeDoux. *The Concentration of Measure Phenomenon*. American Mathematical Society, 2001.

[79] Yuh-Jye Lee and Olvi Mangasarian. RSVM: Reduced support vector machines. In *First SIAM International Conference on Data Mining*, Chicago, 2001.

[80] Kobi Levi and Yair Weiss. Learning object detection from a small number of examples: The importance of good features. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, volume 2, pages 53–60, Washington, DC, 2004.

[81] Stan Z. Li, ZhenQiu Zhang, Heung-Yeung Shum, and HongJiang Zhang. FloatBoost learning for classification. In *Advances in Neural Information Processing Systems*, 2002.

[82] David G. Lowe. *Perceptual Organization and Visual Recognition*. Kluwer Academic Publishers, Boston, MA, 1985.

[83] David G. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(5):441–450, May 1991.

[84] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int'l Journal of Computer Vision*, 60(2):91–110, November 2004.

[85] Bruce Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[86] John MacCormick and Andrew Blake. A probabilistic exclusion principle for tracking multiple objects. In *Proc. Int'l Conf. Computer Vision*, pages 572–578, 1999.

[87] L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing-time and memory requirements. *Pattern Recognition Letters*, 15(1):9–17, January 1994.

[88] D. M. Mount and S. Arya. Ann: Library for approximate nearest neighbor searching. In *Proc. Center for Geometric Computing Second Ann. Fall Workshop Computational Geometry*, 1997.

[89] Kenji Okuma, Ali Taleghani, Nando de Freitas, James J. Little, and David G. Lowe. A boosted particle filter: Multitarget detection and tracking. In *Proc. European Conf. Computer Vision*, volume 1, pages 28–39, 2004.

[90] Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[91] Emanuel Parzen. On estimation of a probability density function and mode. *Ann. Math. Stat.*, 33(3):1065–1076, 1962.

[92] P. Pérez, C.Hue, J. Vermaak, and M. Gangnet. Color-based probabilistic tracking. In *Proc. European Conf. Computer Vision*, pages 661–675, 2002.

[93] Vasanth Philomin, Ramani Duraiswami, and Larry Davis. Quasi-random sampling for condensation. In *Proc. European Conf. Computer Vision*, volume 2, pages 134–149, 2000.

[94] Michael K. Pitt and Neil Shephard. Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association*, 94(446):590–599, 1999.

[95] Tomaso Poggio and Federico Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247:978–982, 1990.

[96] Tomaso Poggio and Steve Smale. The mathematics of learning: Dealing with data. *Notices of the American Mathematical Society (AMS)*, 50(5):537–544, 2003.

[97] J. G. Postaire and C. Vasseur. A fast algorithm for nonparametric probability density estimation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 4(6):663–666, 1982.

[98] J. Puzicha, J. Buhmann, Y. Rubner, and C. Tomasi. Emperical evaluation of dissimilarity measures for color and texture. In *Proc. Int'l Conf. Computer Vision*, pages 1165–1172, Kerkyra, Greece, 1999.

[99] Anand Rangarajan, Haili Chui, and Fred L. Bookstein. The softassign procrustes matching algorithm. In *Proc. of the 15th International Conference on Information Processing in Medical Imaging*, pages 29–42. Springer-Verlag, 1997.

[100] Vikas Raykar, Ramani Duraiswami, and Changjiang Yang. The improved fast Gauss transform for variable bandwidth Gaussian kernel machines. Submitted to NIPS, 2005.

[101] Ryan Rifkin. *Everything Old Is New Again: A Fresh Look at Historical Approaches in Machine Learning*. PhD thesis, MIT, Cambridge, MA, 2002.

[102] Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *Ann. Math. Stat.*, 27(3):832–837, 1956.

[103] Henry Schneiderman. Feature-centric evaluation for efficient cascaded object detection. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, volume 2, pages 29–36, Washington, DC, 2004.

[104] Bernhard Schölkopf and Alexander Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, Cambridge, MA, 2002.

[105] Gregory Shakhnarovich, Paul Viola, and Trevor Darrell. Fast pose estimation with parameter-sensitive hashing. In *Proc. Int'l Conf. Computer Vision*, volume 2, pages 750–757, Nice, France, 2003.

[106] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000.

[107] Jianbo Shi and Carlo Tomasi. Good features to track. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, pages 593–600, Seattle, WA, 1994.

[108] B. W. Silverman. Algorithm AS 176: Kernel density estimation using the fast Fourier transform. *Appl. Stat.*, 31(1):93–99, 1982.

[109] Cristian Sminchisescu and Bill Triggs. Kinematic jump processes for monocular 3D human tracking. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, volume I, pages 69–76, Madison, WI, 2003.

[110] Alex Smola and Peter Bartlett. Sparse greedy Gaussian process regression. In *Advances in Neural Information Processing Systems*, pages 619–625. MIT Press, 2001.

[111] Alex Smola and Bernhard Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proc. Int'l Conf. Machine Learning*, pages 911–918. Morgan Kaufmann, 2000.

[112] J. Strain. The fast Gauss transform with variable scales. *SIAM J. Sci. Statist. Comput.*, 12(5), 1991.

[113] Xiaobai Sun and Nikos P. Pitsianis. A matrix version of the fast multipole method. *SIAM Review*, 43(2):289–300, 2001.

[114] J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[115] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *Proc. Int'l Conf. Computer Vision*, pages 839 – 846, Bombay, India, 1998.

[116] Emanuele Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice-Hall, Upper Saddle River, NJ, 1998.

[117] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.

[118] Vladimir Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[119] Paul Viola and William M. Wells III. Alignment by maximization of mutual information. *Int'l Journal of Computer Vision*, 24(2):137–154, 1997.

[120] Paul Viola and Michael J. Jones. Robust real-time face detection. *Int'l Journal of Computer Vision*, 52(2):137–154, May 2004.

[121] Grace Wahba. *Spline Models for Observational Data*. SIAM, Philadelphia, PA, 1990.

[122] Andrew R. Webb. *Statistical Pattern Recognition*. John Weley & Sons, UK, 2nd edition, 2002.

[123] Christopher K. I. Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, pages 682–688. MIT Press, 2001.

[124] Christopher K.I. Williams and David Barber. Bayesian classification with Gaussian processes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(12):1342–1351, December 1998.

[125] Donna J. Williams and Mubarak Shah. A fast algorithm for active contours and curvature estimation. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 55(1):14–26, January 1992.

[126] Oliver Williams, Andrew Blake, and Roberto Cipolla. A sparse probabilistic learning algorithm for real-time tracking. In *Proc. Int'l Conf. Computer Vision*, pages 353–360, Nice, France, 2003.

[127] Christopher Richard Wren, Ali Azarbayejani, Trevor Darrell, and Alex Pentland. Pfinder: Real-time tracking of the human body. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(7):780–785, 1997.

[128] Ying Wu. Robust visual tracking by integrating multiple cues based on co-inference learning. *Int'l Journal of Computer Vision*, 58(1):55–71, June 2004.

[129] Changjiang Yang, Ramani Duraiswami, and Larry Davis. Efficient kernel machines using the improved fast Gauss transform. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1561–1568. MIT Press, Cambridge, MA, 2005.

[130] Changjiang Yang, Ramani Duraiswami, and Larry Davis. Efficient mean-shift tracking via a new similarity measure. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, volume I, pages 176–183, San Diego, CA, June 2005.

[131] Changjiang Yang, Ramani Duraiswami, and Larry Davis. Fast multiple object tracking via a hierarchical particle filter. In *Proc. Int'l Conf. Computer Vision*, Beijing, China, October 2005.

[132] Changjiang Yang, Ramani Duraiswami, Ahmed Elgammal, and Larry Davis. Real-time kernel-based tracking in joint feature-spatial spaces. Submitted to CVPR2004, 2003.

[133] Changjiang Yang, Ramani Duraiswami, Nail Gumerov, and Larry Davis. Improved fast Gauss transform and efficient kernel density estimation. In *Proc. Int'l Conf. Computer Vision*, pages 464–471, Nice, France, October 2003.

[134] Changjiang Yang, Ramani Duraiswami, and Nail A. Gumerov. Improved fast Gauss transform. Technical Report CS-TR-4495, UMIACS, University of Maryland, College Park, MD, 2003.

[135] Jie Yang and Alex Waibel. A real-time face tracker. In *Proceedings of WACV*, pages 142–147, Sarasota, FL, 1996.

[136] Ming-Hsuan Yang, Dan Roth, and Narendra Ahuja. A SNoW-based face detector. In *Advances in Neural Information Processing Systems*, pages 862–868, 1999.

[137] Ting Yu and Ying Wu. Collaborative tracking of multiple targets. In *Proc. IEEE Conf. Comp. Vision Pattern Recognition*, volume I, pages 834–841, Washington DC, 2004.