# MATCHING AFFINE-DISTORTED IMAGES

A Dissertation Presented

by

RAGHAVAN MANMATHA

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 1997

Department of Computer Science

# MATCHING AFFINE-DISTORTED IMAGES

A Dissertation Presented

by

RAGHAVAN MANMATHA

Approved as to style and content by:

Allen R. Hanson, Chair

John Oliensis, Member

Robin J. Popplestone, Member

Edward M. Riseman, Member

James R. Bergen, Member

David W. Stemple, Department Chair
Department of Computer Science

*To my parents S. Raghavan and R. Lakshmi*

# ACKNOWLEDGEMENTS

My dissertation would not have been possible but for the help I received from various people. Al Hanson and Ed Riseman provided continued encouragement, advice and support through the course of this research. I wish to thank John Oliensis for supporting me part of the time and for patiently reading many early drafts of the dissertation and for the many constructive criticisms he gave. Chapter 6 was substantially revised with the help of the committee's comments, in particular Jim Bergen's comments. I also wish to think Robin Popplestone for serving on my committee and for interesting discussions.

The Umass vision group that I joined was an exciting place to be and it was fun to interact with the various people in the group. My thanks go out to the various people in the group with whom I have had the fortune to interact. In particular, I would like to thank Harpreet Sawhney, Rakesh (Teddy) Kumar, Lance Williams and Brian Burns for stimulating discussions on vision and life. Harpreet deserves thanks for inviting me to come to IBM Almaden and for discussions on my dissertation and other aspects of vision and life. The visit to Almaden lead me to explore my current area of research - image retrieval. My interaction with Teddy lead me to start writing short stories for a while. I enjoyed this activity immensely and have to thank Teddy for it. I enjoyed arguing with Lance about both vision and dinosaurs. Our arguments were passionate and fun. Brian was available to discuss any topic on the face of the earth - a rarity in this age of specialization. Others who contributed to this stimulating environment included Chris Connolly, Zhongfei Zhang, Yong Cheng, Bob Collins, Ross Beveridge and Bruce Draper.

Bob Heller has for years been a friendly presence in the laboratory always ready to give advice on systems work. Jonathan Lim also helped me with the systems work in addition to helping me take some of the pictures in Chapter 6. David Hirvonen helped run some of

the experiments in Chapter 7. Thanks to Laurie Downey and Janet Turnbull for providing excellent adminstrative support.

In recent years, I have been working on multi-media indexing and retrieval and have come to interact with a number of people. I thank Bruce Croft for providing me with the opportunity to work in this area. Srinivas Ravela and I have enjoyed a stimulating partnership working on image retrieval. Victor Wu and I have collaborated on retrieving text from images. And of course Adam Jenkins for providing systems support

# ABSTRACT

## MATCHING AFFINE-DISTORTED IMAGES

SEPTEMBER 1997

RAGHAVAN MANMATHA

B. Tech., INDIAN INSITUTE OF TECHNOLOGY, KANPUR

M.S., UNIVERSITY OF HAWAII

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Allen R. Hanson

Many visual tasks involve the matching of image patches derived from imaging a scene from different viewpoints. Matching two image patches can, however, be a difficult task. This is because changes in the relative orientation of a surface with respect to a camera cause deformations in the image of the surface. Thus this deformation needs to be taken into account when matching or registering two image patches of an object under changes in viewpoint. Up to first order these deformations can be described using an affine transform.

Here, a computational scheme to match two image patches under an affine transform is presented. The two image patches are filtered with Gaussian and derivative of Gaussian filters. The problem of matching the two image patches is then recast as one of finding the amount by which these filters must be deformed so that the filtered outputs from the two images are equal. For robustness, it is necessary to use the filter outputs from many points in a small region to obtain an overconstrained system of equations. The resulting equations are linearized with respect to the affine transforms and then iteratively solved for the affine

transforms. The method is local and can match image patches in situations where other algorithms fail. It is also shown that the same framework may be used to match points and lines.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

Many visual tasks involve the matching of image patches derived from imaging a scene from different viewpoints. The structure–from motion–problem involves matching two or more image patches arising from a single surface patch seen from different viewpoints. The shape–from–texture problem may be viewed as involving the matching of two or more image patches arising from viewing different surface patches with similar textures. Some applications are concerned primarily with matching image patches without requiring explicit inferences about the imaged objects. These include the registration of video or medical images [68, 56], mosaicing [25, 68], object recognition [71] and the retrieval of images based on their similarity to other images in a database [47].

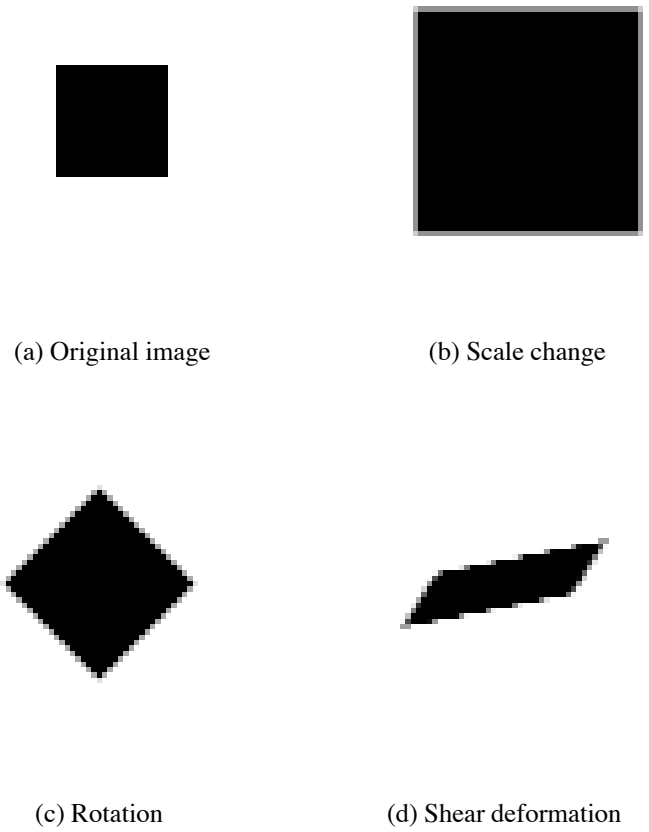
There are two different but closely related notions of matching implicit in the above applications. One view of matching is that it involves the recovery of the correct transformation so that two image patches map to each other. A second, more qualitative, view of matching is that it may be used to verify whether two images are of the same object [71].

Matching two image patches can be a difficult task. This is because changes in the relative orientation of a surface with respect to a camera cause deformations in the image of the surface. Thus this deformation needs to be taken into account when matching or registering two image patches of an object under changes in viewpoint. Up to first order, there are four kinds of image deformations - a scale change, rotation in the image plane and shears along the x and y axes. The position of the surface's projection in the image also

changes with motion. To a first order approximation, this image translation together with the deformation can be described using a six parameter affine transformation ($\mathbf{t}$ , $\mathbf{A}$ ) where

$$\mathbf{r}' = \mathbf{t} + \mathbf{A}\mathbf{r} \tag{1.1}$$

$\mathbf{r}$' and $\mathbf{r}$ are the image coordinates related by an affine transform, $\mathbf{t}$ is a 2 by 1 vector representing the translation, and $\mathbf{A}$ is the 2 by 2 affine deformation matrix. Figure 1.1 shows the different kinds of deformations possible. Scaling the black square in Figure 1.1(a) gives give Figures 1.1(b), rotating the square produces Figure 1.1(c) and shearing it produces Figure 1.1(d). This deformation may also be thought of as geometric distortion,



(a) Original image (b) Scale change

(c) Rotation (d) Shear deformation

**Figure 1.1.** Varieties of deformations

since it is a deformation of the underlying coordinate system. Thus, for example, under and affine transformation, a circle will be distorted or deformed into an ellipse.

Although the deformations make matching difficult, they may be used to infer local surface geometry and depth from motion. Since a repeating texture pattern can be thought of as a pattern in motion, shape from texture can also be derived from deformations [46].

For many applications, it suffices to match image patches assuming affine deformations for the following reason. It can be shown that under orthography or weak perspective projection two views of a planar patch differ by a 6 parameter affine transform [30]. Although for more general projection models and more general surfaces the transformation between two views is not affine, a local affine approximation suffices for many situations. This is because scenes typically contain smooth objects with a finite number of discontinuities. When the curvature is not too great, smooth objects can be parametrized locally by a planar model. If the patches are sufficiently small, the resulting local affine transformation between image patches often suffices for matching.

The difficulty of matching two image patches which differ by an affine transform is illustrated by Figure (1.2). The image on the right is scaled 1.4 times the image on the left. Traditional matching uses fixed correlation windows or filters and, therefore, has a problem with matching differently sized images such as the ones shown in Figure (1.2). Thus, Washington's face, for example, in the two pictures cannot be matched by a traditional scheme. The correct way to approach this problem is to deform the correlation window or filter according to the image deformation.[1] For a general affine transform the deformations are even more extreme.

This dissertation derives a computational scheme to match two image patches under an affine transform. The two image patches are filtered with Gaussian and derivative of Gaussian filters. The problem of matching the two image patches is then recast as one of

[1]This discussion applies to image or filter based matching schemes and not to schemes based on matching geometric tokens.

finding the amount by which these filters must be deformed so that the filtered outputs from the two images are equal. For robustness, it is necessary to use the filter outputs from many points in a small region to obtain an overconstrained system of equations. The resulting equations are linearized with respect to the affine transforms and then iteratively solved for the affine transforms. The method is local and can match image patches in situations where other algorithms fail.

Experiments are shown on both random dot and real images. One of the algorithms devised here is also applied to the problem of word spotting, that is, indexing hand writing by matching word images [45, 44]. It is shown that the algorithm performs better on standard information retrieval measures like recall and precision than other algorithms tried on this problem. Other researchers [51] have recovered image blur and the affine transform from a pair of blurred images of a scene using one of the algorithms derived in this study.

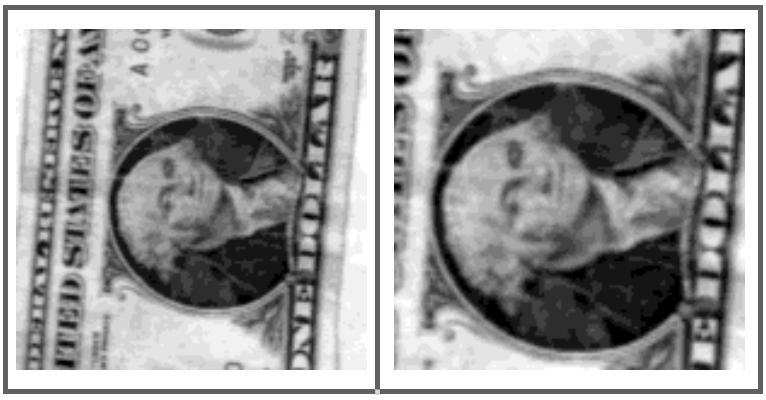


**Figure 1.2.** Dollar bill scaled 1.4 times

## 1.1 Applications of Affine Matching

There are many practical situations in which it is desirable to use affine matching or a subset of it. In recent years, the problem of registering two or more images to create

larger mosaics has become important [25, 68]. One approach to this problem is to match the images assuming an affine transform has occured - a good approach provided that the scenes which are imaged are approximately planar. For more complicated scene structures, the planar parallax [58, 4] approach has been used. The planar parallax approach involves computing an affine transform between two images and then computing the residual flow required to further match the two images. Thus an integral part of the planar parallax approach is the computation of an affine transform between two images.

A related problem is the registeration of medical images. Sometimes, the medical images to be registered are two dimensional in nature. Often, the images are three dimensional and are matched using three dimensional affine transforms.

Affine transforms have also been used to compute structure from motion. In some situations, the motion can be approximated using a similarity transform (a special case of an affine transform with only 4 parameters). For example, when a robot's motion is mostly translation, then the changes in image projections of shallow structures (i.e. structures whose extent in depth is small compared to their distance from the camera) may be described using a similarity transform [57]. In other cases, the motion and structure of a surface may often be described using an affine transform or a set of affine transforms.

The problem of deriving shape–from–texture may be viewed as analogous to the structure from motion problem. In both cases, shape may be recovered by matching corresponding image patches. Many shape–from–texture techniques are based upon recovering the affine transform or a subgroup of an affine transform between two image patches [].

Certain approaches to object recognition use affine transformations to recognize objects by matching images to object models under an affine transformation. For example, in the alignment approach, objects are recognized by checking whether a model object can be aligned with the image using one or more planar surfaces on the model [24]. The alignment approach is based on the assumption that the transformation between the image and model can be described using an affine. Ullman [71] has argued that objects can be recognized

by using a set of view variant models and matching these to an image assuming that the transformation is locally affine.

A related problem is the retrieval of images from a database, based on their visual similarity to a given image. Similarity transforms can also be used for the problem of matching logos or trademarks in a database. Often such logos or trademarks differ by a large scale change and a translation which can be modelled by a similarity transform. The preparation of an index for handwritten archival manuscripts (also termed word spotting [45, 44] has also been formulated as a matching problem. Here, the images of words are matched assuming an affine transform and then ranked. One of the algorithms created here performs well when applied to the problem of word spotting (see chapter 7).

## 1.2 Framework

In this dissertation, matching is done by comparing the outputs of a filters applied to the two image patches. Under an affine transform the shape and size of the two patches is different. The change in shape and size of the patches is handled by deforming the filters appropriately. Previous methods have not handled the change in shape and size of the image patches correctly [6, 28]. If a Gaussian (or Gaussian derivative) is used as a filter, the problem of measuring the affine deformation may be recast into the problem of finding the deformation parameters of the Gaussian (or Gaussian derivative). For example, let $F_1$ and $F_2$ be two images related by a scale change s. Then the output of $F_1$ filtered with a Gaussian of $\sigma$ will be equal to the output of $F_2$ filtered with a Gaussian of $s\sigma$. Similar relationships (equations) hold for arbitrary affine transforms and filters described by derivatives of Gaussians. These equations are exact for any arbitrary affine transform in arbitrary dimensions.

In two dimensions, the required deformation is simple to describe. An affine transform maps a circle to an ellipse. Since the level curves of a Gaussian are circular, the level curves of the deformed Gaussian must be elliptical giving rise to elliptical Gaussians. Let

$F_1$ and $F_2$ be two–dimensional images related by an affine deformation. Then the output of $F_1$ filtered with an Gaussian of scale $\sigma$ will be equal to the output of $F_2$ filtered with an elliptical Gaussian of whose parameters are defined by the matrix $\mathbf{A}\mathbf{A}^t\sigma^2$ (for a definition of an elliptical Gaussian see chapter 3).

The problem of matching the two image patches has, therefore, been reduced to the problem of finding the affine parameters of the deformed Gaussian with which to filter the second image. A brute force approach to recovering the affine parameters is to sample the space of affine parameters and filter the images with Gaussians using the sampled parameters. The correct affine transform will minimize the difference between the two images. This is computationally very expensive.

The alternative proposed here is to coarsely sample the affine space and then use an iterative procedure to solve for the affine transform. The Gaussians (or Gaussian derivatives) are linearized with respect to the affine parameters. Gaussians and Gaussian derivatives are known functions. Therefore, linearizing them is much simpler to do than linearizing the images directly. The resulting linear equations maybe cast as a linear–least–mean squares problem. A solution to the linearized equations is obtained and the first image patch warped using this solution. The affine transform is then computed by using the warped first image patch and the second image patch. This procedure is repeated for a number of iterations. No other techniques follow this approach.

Such linearized equations can be derived at every point (pixel) where an image is filtered with a Gaussian or Gaussian derivative. Additional equations can also be obtained by using filters at different scales. These equations can be combined in different ways to create a number of different algorithms for solving the affine transforms and for matching the image patches.

For the special case when the affine transform is a similarity transform, it can be solved for by filtering the image patches with a Gaussian at a single point at multiple scales. The general affine transform cannot be recovered in this way. It turns out that even if the

Gaussian and its first and second derivatives are used (an approach used by Werkhoven and Koenderink [75]), the results obtained in the case of the general affine transform are poor. Instead in our approach, the filter outputs from different points in a small region are pooled to create systems of equations which can be solved for the general affine transform. Different algorithms may be formulated depending on the filter used.

Two algorithms are constructed. The first, based on filtering with the Gaussian at a number of points in a region is called GauArea. The second, based on filtering with the first derivatives of a Gaussian at a number of points in a region and is called DGauArea. Experiments are shown for a representative set of scale changes, rotations and shears for both algorithms. For comparison purposes, two other algorithms GauAreaN and DGauAreaN are also developed and applied to the same data. GauAreaN and DGauAreaN are similar to GauArea and DGauAreaN but the effects of the geometric distortion on the filters are not taken into account. It is seen that usually there are significant advantages when the geometric distortion of the filters is accounted for.

It is also shown that images made of point and line tokens can be treated as part of the same framework. For point and line tokens, explicit correspondence need not be established. Further, the method is applicable for both closed and open contours and even for collections of line segments.

The remaining chapters are organized as follows. Chapter 2 reviews related work on measuring affine transforms. Chapter 3 formulates the problem of measuring affine transforms in images. Chapter 4 solves for similarity transforms. Chapter 5 derives two algorithms GauArea and DGauArea for solving for general affine transforms and also two algorithms GauAreaN and DGauAreaN which do not account for the deformations of the Gaussians. In Chapter 6, experimental results are shown on these algorithms for both random dot and real images. Chapter 7 proposes some applications of the techniques for solving affine transforms. Chapter 8 shows how the methods can be modified for use with point and line tokens. In Chapter 9, the conclusions derived from this study are stated. A

set of appendices also follows the main body of the work. In Appendix A, certain results used in Chapter 5 are derived. Appendix B discusses how to compute large translations. Appendix C contains a more detailed set of graphs for the experiments in Chapter 6.

# CHAPTER 2

# REVIEW OF TECHNIQUES FOR RECOVERING IMAGE DEFORMATION

Matching can be based either on tokens or on image features. Tokens will be defined here as geometric structures in the image like points, straight or curved lines or contours. Usually there is assumed to be a one to one correspondence between tokens and geometric structures in the world. e.g. it is assumed that a line token corresponds to a "physical" line in the scene. This correspondence between tokens and scene elements ensures that the tokens will be stable over a sequence of images. In the ideal case (i.e. without noise or occlusion), token matching can be done by establishing a one to one correspondence between tokens from two images.

Image features will be defined as functions computed over the image brightness. Usually image features are computed as the outputs of linear filters applied to the image. Feature based methods do not make any assumptions about the presence of any specific structures in the image and there is no implicit assumption that the feature corresponds to a geometrical structure in the scene. A feature based matching technique finds that transformation which maps the image features in one image to those in the second image. The simplest case is where the brightnesses in one image are mapped to those in the second image. Both (image) feature based and token based methods have their distinct advantages. Techniques which incorporate both image features and tokens would be ideal, but none have so far been proposed.

The categorization of matching techniques into token based and feature based methods is convenient because with one exception, token based and feature based techniques are

different. The exception is techniques based on computing moments over the image which can be applied to both image features and certain types of tokens (moment based techniques find that transformation which map the moments computed from one image to the moments computed from the second image). Token based techniques will, therefore be reviewed first. This will be followed by a review of methods based on moments. Finally, a review of approaches based on image features will be undertaken. Rather than being comprehensive, the review here will attempt to elucidate principles and issues by focusing on certain key ideas. For a more exhaustive listing of papers in the area of registration and matching see [10, 55].

## 2.1 Token Based Methods

Token based methods have a number of advantages over techniques based on image features. Tokens are often sparsely distributed and in general this leads to a reduction in the amount of data handled and consequently the computation time. If corresponding tokens can be identified, token based algorithms may be used to compute even large affine transforms which are otherwise difficult to recover.

To establish effective correspondence, it is important to ensure that the tokens detected are stable over a sequence of images. Early methods to find token points used interest operators to detect them []. The problem with such an approach was that the token points were typically not stable, and hence correspondence could not be effectively established. Other techniques attempted to find tokens that corresponded to physical structures in the scene [49]. The assumption was that such tokens would be much more stable. For example, junction and corner detectors were devised to find point tokens coresponding to junctions and corners in the scene. Similarly, line detectors were devised to find line tokens which corresponded to the boundary lines between surfaces in the scene. However, algorithms which detect tokens recover image structures and not scene structures, so that the assumption that tokens would be stable may not always be valid. In addition, the shape of the image struc-

ture changes with viewing geometry, and token detectors cannot therefore always locate tokens accurately under large viewpoint changes.

There are two main strategies for finding correspondence in token matching problems. The first strategy assumes that correspondence can be independently recovered and the transformation can then be computed from it. Correspondence is often established by tracking tokens over a series of closely spaced frames. The small motion between adjacent frames implies that token locations do not change significantly between them. If, in addition, the tokens are sparsely distributed and distinctive their correspondence can then be easily established. Cipolla and Blake [13], for example, track a closed contour over a set of closely spaced frames to establish correspondence. An alternative strategy is to compute both the correspondence and the deformation simultaneously [59, 63, 61]. Sawhney [59], for example, tracks sets of 3 lines by requiring that the tracked sets be consistent with a similarity transform over many frames.

Given sets of corresponding tokens from two images, the affine transformation is given by the coordinate transformation between the locations of corresponding sets of tokens. For example, the locations of three (token) points in two images suffice to determine an affine transformation between them. Robustness against noise and token localization errors may be achieved by using a least squares or a least-median squares fit to a large number of tokens. Given reliable token correspondences, the deformation can be recovered rapidly and accurately. Cipolla and Blake [13] compute the afffine transform from the area change of the closed contours of the extreme frames.

The alternative strategy of computing correspondence and the transformation simultaneously has been recently explored by a number of researchers. Sawhney's technique [59], mentioned above, requires that the frames be closely spaced and that the lines be tracked over at least three frames. A recent class of methods based on using point tokens has approached the problem from a different point of view. These techniques work with just two frames. They assume that tokens have been extracted previously, but try to capture the

uncertainty in token localization by using a distance measure between Gaussian blurred images [63, 61]; we also propose such an approach in Section 8.1. The techniques may be viewed as similar to correlation matching and also to some filter based techniques (which will be discussed later). These two-frame techniques are worth discussing in some detail.

Scott and Longuet Higgins [61] proposed an algorithm to recover both an affine transform and the correspondence simultaneously between two set of points, I from the first image and J from the second image. They first computed an adjacency matrix **G**. The entries $G_{ij}$ are Gaussian weighted distances between a point i in set I and a point j in set J. Each entry $G_{ij}$ is given by

$$G_{i,j} = exp(-r_{ij}^T r_{ij}/(2\sigma^2)) \tag{2.1}$$

where $r_{ij}$ is the Euclidean distance between i and j. The matrix **G** is then diagonalized using singular value decomposition (SVD) to give

$$\mathbf{G} = \mathbf{TDU} \tag{2.2}$$

where **D** is a diagonal matrix and T and U are orthogonal matrices. The diagonal entries in **D** are replaced by 1's to give an m by n matrix **E**. The pairing matrix **P**

$$\mathbf{P} = \mathbf{TEU} \tag{2.3}$$

indicates the strength of the attraction between points i and j. Thus a correspondence between two points i and j is posited only if the entry $P_{ij}$ is large. Intuitively, P is the matrix which correlates best with the G matrix in the sense of maximizing the trace of $P^T G$. The transformation can then be computed using the recovered correspondence. Scott and Longuet-Higgins showed that if $\sigma$ is chosen large enough , the method would compute the correspondence correctly for translations, scale changes (i.e. expansions, contractions) and shears. Here, as in intensity based algorithms, the large values of sigma are useful

in recovering large translations. However, the method cannot be shown to compute the correct correspondence if a rotation is involved (in general no technique based on token points can be "proved" to handle rotation because in certain pathological situations, the rotation cannot be recovered). In practice, small rotations can be handled most of the time.

The method was modified by Shapiro and Brady [63] to cope with large rotations (up to 80 degrees) in addition to translation, shear and scale change. This was done by computing distances between points in the same image rather rather than between points in different images. A matrix $H_1$ whose entries are Gaussian weighted distances between points in the first set I is computed. A matrix $H_2$ is similarly computed for points from the second set J. Each $\mathbf{H}$ matrix is diagonalized

$$\mathbf{H} = \mathbf{V}\mathbf{\Sigma}\mathbf{V^T} \tag{2.4}$$

where $\mathbf{V}$ is an orthogonal matrix and $\mathbf{\Sigma}$ is a diagonal matrix. Each row of $\mathbf{V}$ may be regarded as a feature. Feature i is therefore represented by the vector $g_i$. The Euclidean distance between the feature sets $g_{1i}$ and $g_{2j}$ corresponding to the ith row in $\mathbf{V_1}$ and the jth row in $\mathbf{V_2}$ is computed i.e.

$$Z_{ij} = ||g_{1i} - g_{2j}||^2 \tag{2.5}$$

A good match occurs if this distance is small. If the smallest value in row i is the entry $Z_{ij}$ then feature i in the first image corresponds to feature j in the second image i.e. a correspondence between the ith point in the first image and the jth point in the second image is established. If one of the $V$ matrices has more features (the number of features depends on the number of points), the extra features associated with the least significant eigenvalues are deleted. Due to the deletion of extra modes, the algorithm does not perform well if there are too many extra points [63].

Gold et al [18, 38] independently developed a similar algorithm to compute 2D and 3D pose. Intuitively, the main difference in their algorithm is that they allow for the possibility of fuzzy matches. Each entry in the adjacency matrix initially encodes the Gaussian weighted distance between a token point in one image and a token point in the second im-

age. If there are m tokens in the first image and n in the second image, the adjacency matrix is of size m + 1 by n + 1. The additional row and column are used for slack variables. At each iteration the correspondence is first established and the affine transform then computed. The token points are then warped according to the affine transform and the process is repeated. Correspondence is established in the following manner. The constraint that a point in the first image can only match a point in the second image is enforced. For each row this is done by summing the first m numbers in that row and then normalizing it so that the sum equals 1. The process is repeated for all the columns. The entry in a given row with the highest value is then set to 1 (and similarly with the columns). Points which do not have matches will have a 1 in either the m + 1 row or the n + 1 column. In this manner the authors claim to address the problem of occluding points.

The main difficulty with token based schemes is that reliable token correspondences may not always be available. Correspondence between tokens is often established by assigning that token in the second image to the token in the first image whose location is closest to it. Since most tokens are non-distinctive, if more than one token is present in the neighborhood it can make finding correspondence between tokens difficult. In many situations tokens are recovered by techniques which make specific assumptions about images that may not hold in general. For example, both Cipolla and Blake [13] and Sawhney [59][1] assume that objects have large homogeneous planar regions. These techniques would, therefore, fail if the object being tracked does not have any large homogeneous regions which are planar.

A number of other problems can also arise with token based methods. Token localization is often poor. For example, token points are often detected using junction or corner detectors. However, the locations of junction or corner tokens cannot always be determined accurately. In addition, most images contain information over much of the image, not just

[1] Again although Sawhney [59] did not make this assumption explicit, all his experiments were conducted using such objects.

at token locations. Tokens thus do not take advantage of all the information present in an image. Finally, a token present in one frame may be absent in the second due to occlusion. Most token based schemes do not handle this well.

## 2.2 Moment Based Techiques

An early approach to computing affine transforms was based on computing moments over tokens or image features. Since moments are computed by integrating over many image points, (for example, the zeroth order moment of brightness is just the average brightness of the region), they are likely to be fairly robust to noise. In addition, given two sets of tokens or two images related by an affine transform, their moments are related by functions which are polynomial in the affine parameters. For example, the first moments are related by

$$m_1 = \mathbf{A}^T {m'}_1 - t{m'}_0 \tag{2.6}$$

where $m_i$ denotes the $i_{th}$ moment computed over the first image, the ' denotes the same quantity computed over the second image and $\mathbf{t}, \mathbf{A}$ is the affine transform between the two images. Moments also possess the unique property that affine invariants can be constructed using them [55]. Moment techniques require that the region of integration in the second image corresponding to that in the first image be identified apriori. In addition, the higher moments weight the outer regions of the patch more than the interior. If the boundaries of the patch are not known in advance, - which is the situation in most matching problems - this disproptionate weighting can cause points outside the patch to be weighted heavily and thus cause erroneous results. When the region of integration is unknown, this forces the moment methods to be global in nature i.e. they are applied to the entire image.

Hu [22] showed that invariants to affine transforms could be constructed using the theory of algebriac invariants. We will follow Reiss [55] in explaining this technique. Define the moment of an image f(x,y) by

$$m_{pq} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p y^q f(x,y)dxdy \qquad p,q = 0,1,... \tag{2.7}$$

and the central moments by

$$\mu_{p,q} = \int_{-\infty}^{\infty} (x-\bar{x})^p (y-\bar{y})^q f(x,y)dxdy, \qquad p,q = 0,1,2... \tag{2.8}$$

where $\bar{x} = m_{10}/m_{00}$ and $\bar{y} = m_{01}/m_{00}$. Define a binary form of order p as

$$f_p(x,y) = a_{p,0}x^p + \begin{pmatrix} p \\ 1 \end{pmatrix} a_{p-1,1}x^{p-1}y + \begin{pmatrix} p \\ 2 \end{pmatrix} a_{p-2,2}x^{p-2}y^2 + .... + a_{0,p}y^p \tag{2.9}$$

Then there exist functions of the coefficients $\{a_{i,p-i}\}$ called algebraic invariants which are invariant to linear coordinate transformations. A function $I(a_{p,0},...,a_{0,p})$ is called an invariant if

$$I(a'_{p,0},...,a'_{0,p}) = \Delta^g I(a_{p,0},...,a_{0,p}) \tag{2.10}$$

where the ' indicates a coordinate transformation and $\Delta$ is the determinant of the transformation. The precise forms of the algebraic invariants can be found in [55]. Hu showed that if a binary form of order p has an algebraic invariant $I(a_{p,0},...,a_{0,p})$ of weight g and order k, then the central moments of order p have the same invariant with an additional factor i.e.

$$I(\mu'_{p,0},...,\mu'_{0,p}) = \Delta^g |\Delta|^k I(\mu_{p,0},...,\mu_{0,p}). \tag{2.11}$$

An example will clarify Hu's idea. Consider the binary quadratic form

$$f_2(x,y) = ax^2 + 2bxy + cy^2 \tag{2.12}$$

Let there be a linear coordinate transformation of the form

$$\begin{pmatrix} x' & y' \end{pmatrix} = A \begin{pmatrix} x & y \end{pmatrix} \tag{2.13}$$

where A is a 2 by 2 matrix. Then under the coordinate transformation

$$f_2(x', y') = a'x'^2 + 2b'x'y' + c'y'^2 \tag{2.14}$$

for some a',b' and c'. Then it may be shown that a simple invariant is [55]:

$$Q' = a'c' - b'^2 = \Delta^2(ac - b^2)Q \tag{2.15}$$

i.e. Q = I(a,b,c) is an invariant. From Hu's work this implies that the central moments satisfy the following invariant

$$I(\mu'_{20}, \mu'_{11}, \mu'_{02}) = \Delta^2|\Delta|^2 I(\mu_{20}, \mu_{11}, \mu_{02}) \tag{2.16}$$

Moment invariants have also been derived using other techniques. Cyganski and Orr [14] derived moment invariants using tensors. If invariants to rotation are required they can be derived using complex moments [1] while translation invariants can be derived in a straightforward manner from the definition of moments. The rotation invariants may be used to first normalize the images/patterns so that the principal axis has a fixed predetermined orientation. Matching under similarity transforms can, therefore, be carried out by first normalizing with respect to the principal axis and then matching. With modifications, most of these moment based techniques can be applied to points, lines or images. Lee [35], for example, recovers the affine transform given the contours bounding corresponding closed regions.

Essentially all these methods assume that either the pattern has been segmented apriori or that the technique is applied over the whole image. Moment based methods are particularly good for binary images where an implicit segmentation of the image often occurs - the object is white against a dark barkground. For a good review of moment based techniques see [55].

Moment based methods can be used for local patches by using appropriate windows - the windows may also be viewed as weights. However, the generation of algebraic functions which are affine invariant using weighted moments has not been demonstrated and it is not clear that it can be done.

## 2.3 Image Based Methods

Unlike token based techniques, feature based methods take advantage of all the information present in the image. This is because they utilize information everywhere in the image and do not require the existence of specific structures in the image. The disadvantage of methods based on image features is that they are computationally expensive.

Image feature based methods utilize the principle of conservation of brightness between corresponding points in two image patches. This conservation law assumes that changes in brightness due to shading or illumination can be ignored. Let $\mathbf{r}$ be the image point in the first image and $\mathbf{r}_1$ the corresponding point in the second image such that $\mathbf{r}_1 = f(\mathbf{r})$, where f is some coordinate transformation. Then the conservation of brightness [15, 21] is expressed by the following equation,

$$F_1(\mathbf{r}) = F_2(\mathbf{r}_1) \tag{2.17}$$

where $F_1$ is the brightness in the first image and $F_2$ the brightness in the second image.

Constraint equations for different features may be derived from this conservation law in a number of ways. For example, differentiating the above equation gives the following constraint equation

$$dF_1(\mathbf{r})/d(\mathbf{r}) = dF_2(\mathbf{r}_1)/d(\mathbf{r}_1) * df(\mathbf{r}_1)/d(\mathbf{r}) \tag{2.18}$$

In this case, the feature used is the derivative of the brightness. Other example constraint equations may be derived by filtering the image using linear filters. For example, under a

similarity transform ($f = s\mathbf{R}$ where $\mathbf{R}$ is a rotation matrix and s the scale) , filtering the image with a Gaussian gives the following constraint equation:

$$F_1(\mathbf{r}) * G(., \sigma) = F_2(\mathbf{r}_1) * G(., s\sigma) \tag{2.19}$$

where $G(., \sigma)$ is a Gaussian with standard deviation $\sigma$. In this case, the features are the outputs of the filters. Constraint equations can also be derived in the frequency domain by computing the Fourier transform of both sides of the above equation (2.17). The features in this case are the Fourier transformed outputs.

These constraint equations are functions of the transformation parameters and may, therefore, be used to solve for the transformation parameters. For example, under an affine transformation, the following constraint equation can be derived from equation (2.17).

$$F_1(\mathbf{r}) = F_2(\mathbf{A}\mathbf{r} + \mathbf{t}) \tag{2.20}$$

A linear approximation to this equation gives the following brightness constraint equation.

$$F_1(\mathbf{r}) \approx F_2(\mathbf{r}) + (\mathbf{A}\mathbf{r} + \mathbf{t})^T F_2(\mathbf{r}) \tag{2.21}$$

By applying the constraint equation to each pixel in a region R, an overconstrained system of equations can be obtained. The optimal correspondence, in a least squares sense, can then be determined from this system of equations [11].

A "spatial assumption" is made that over some neighborhood the transformation between two patches can be described by an affine. The justification for the spatial assumption follows from the fact that locally the transformation between the different image projections of a smooth surface can be approximated by an affine transform. In such a situation, the affine parameters are likely to vary slowly over a local neighborhood. The spatial constraints are related to smoothness assumptions on the parameters. Smoothness assumptions

on the image flow were typical of early optical flow algorithms e.g. [21, 3]. The "spatial assumption" is more directly based on an approximation to the local surface geoemtry than smoothness assumptions on the optical flow.

Multiple constraint equations can be obtained in two different ways:

1. By using many different features at a given point. For example, filtering the image with a Gaussian and its derivatives at a point gives multiple constraint equations [31, 46, 40, 75].

2. By applying the constraint equation for a given feature to many different points in a neighborhood [6, 11].

These two approaches may also be combined [41].

A number of factors influence the accuracy of the two approaches. To avoid aliasing, the central regions of a filter must be weighted more than the extremities (i.e. the filters must fall off gradually). Thus, most of the information obtained by filtering is from a small area around the center of the region. Since discretization limits the precision with which measurements can be made, affine transforms computed using small areas are a poor approximation to the correct result (for a more detailed discussion, see Section 3.5). By using filters centered at many different points, a much larger area may be used to compute the affine transform thereby improving the accuracy of the computation. Since both aliasing and discretization must be considered when computing any feature, the same results apply to computations of affine transforms using image features. In general, therefore, the affine transform results from the first approach (using multiple features at a single point) are poorer [41, 75] than those using the second approach (constraint equations at multiple points) [6, 41]; even if a number of features are available, it is usually necessary to find constraint equations at many points (this will also be demonstrated in Chapter 5).

The simplest way to compute affine transformations or optic flow is to test for each possible transformation while directly comparing the intensity values of both images. This

is known as area correlation and has led to useful algorithms for finding translation [3]. Area correlation is expensive when more parameters need to be recovered because it tests for each possible deformation while directly comparing the intensity values of both images. If the transformations are small, linearizing the constraint equations with respect to the transformation parameters eliminates the search requirements. The transformation parameters can then be solved directly [11]. This strategy is widely used for computing optic flow. Large translations can be recovered by using a multiscale coarse-to-fine approach. The method can be extended to compute moderate–sized affine transforms by linearizing the brightness around the current estimate of the affine transform: this may be done by first warping one image with respect to the other using the current estimate of the affine transform and then recomputing the affine transform. If the affine parameters include large translations, a multi-scale version of the technique may be used [6]. This approach by Bergen et al [6] leads to good results for moderately sized affine transforms.

There are a number of advantages to using features other than brightness. For example, methods based on using the brightness directly are susceptible to illumination changes and this may be avoided by using other features [34, 75, 27]. Using brightness directly has other disadvantagesas well. Consider the brightness constraint equation applied to a number of pixels in a region. If a least mean squares solution of this set of equations is obtained, pixels which are brighter will be weighted more than the others. However, this is not desirable if there are large homogeneous regions, for such regions provide little information. Instead, it is preferable to weight pixels where the gradient or some other quantity is large. This can be done automatically by using features obtained by filtering the image with, for example, the derivatives of Gaussians. It is also possible to obtain features invariant to particular transformations. For example, features like the power spectral density are invariant to small changes in translation.

By comparing feature values from the two image patches, both the correspondence and the transformation parameters may be recovered. A least means squares solution may be

used to do this in a manner simialr to the brightness case. Features are expensive to compute and so it may be desirable to minimize the number of features used.

Feature set approaches have been used to compute good approximations to stereo correspondence [31], to optical flow, [73] and to affine transforms by searching over the space of features [27]. Small affine transforms have also been computed by linearizing the constraint equation values with respect to the transformation parameters [75, 76].

We now examine some of these methods in greater detail. This will be largely confined to techniques used to measure affine transformations; for a review of techniques to find optic flow see [].

### 2.3.1 Brute Force Search

The most straightforward method to match image patches is to directly correlate the image intensities of a model image patch with another image patch by searching over the space of affine parameters.

Consider an image patch $F_2$ obtained by affine transforming the patch $F_1$. i.e.

$$F_1(\mathbf{r}) = F_2(\mathbf{A}\mathbf{r} + \mathbf{t}) \tag{2.22}$$

where ($\mathbf{t}$,$\mathbf{A}$) is the affine transform, $\mathbf{t}$ being the translation and $\mathbf{A}$ the 2 by 2 affine deformation matrix.

An initial correspondence is obtained by correlating the two image patches while assuming that there is no deformation ie. $\mathbf{A} = \mathbf{I}$. This solves for the translation parameters and corresponds to the case where the optical flow is computed [3].

The simplest way to recover the affine transform is to sample the space of affine transforms so that the equality in equation 2.22 is satisfied for the two patches. Consider two corresponding regions $R_1$ and $R_2$ in the two images. The pixel intensities in $R_1$ and $R_2$

may be written in the form of column vectors $\mathbf{I}_1$ and $\mathbf{I}_2$. These intensities are then related by the matrix $\mathbf{T}$ which is a function of the affine parameters.

$$\mathbf{I}_1 = \mathbf{T}\mathbf{I}_2 \tag{2.23}$$

The space of affine transforms is sampled and $\mathbf{T}$ precomputed for each set of sample values. The affine parameters for which the above equation is best satisfied in a least squares sense is selected.

Instead of using all the pixel values, Jones and Malik [28, 27] used a basis set. They filtered the regions $R_1$ and $R_2$ with a set of Gaussians and derivatives of Gaussians at several different scales. The outputs of the filters were then stored in the column vectors $\mathbf{I}_1$ and $\mathbf{I}_2$. The resulting column vectors are smaller than if all the pixels in the entire image were used. Initial correspondence was again obtained as before by assuming that there is no deformation and minimizing the following error function.

$$|\mathbf{I}_1 - \mathbf{I}_2|^2 \tag{2.24}$$

There are a number of problems with the above approach. First, this approach involves a brute force search through the space of affine parameters and is therefore very expensive. Second, the regions $R_1$ and $R_2$ are of the same size, since the error function compares pairs of pixels one from each window. Image patches can usually undergo deformations due to affine transforms. Under a large affine transform, the size of one patch may be significantly different from that of the other. A method which assumes that the two patches are roughly the same size will, therefore, work only for small affine transforms. Alternatively, one could compare an image patch with a deformed version of the second image patch, warping the second image patch for every possible sample in the parameter space, which would be computationally very expensive. The method used to find initial correspondence

(equation(2.24)) also assumes small affine transforms (for example it would fail if there was a large rotation).

In the special case of stereo the search is more manageable although still very expensive. This is because changes in the vertical disparity parameters may be assumed to be zero (i.e. the affine parameters $a_{21} = 0$ and $a_{22} = 0$) for the standard stereo configuration where the cameras are verged by moving them in the epipolar plane. Since the relative positions of the camera are known, the problem of computing the translation is simplified to a 1-D search along the epipolar line. Further, in many situations the affine transform can also be assumed to be small. Jones and Malik [28, 27], therefore, employed the technique of using a basis set with exhaustive search (see equation 2.24) only for the case of stereo vision. A total of 70 features were employed by using Gaussian derivatives up to third order at seven different scales.

### 2.3.2 Linearized Methods

Due to the complexity and problems inherent in the brute force approaches, most techniques to measure affine transforms start with the basic equations 2.20 and then linearize them. Linearization leads to a set of linear equations which can be solved for the affine parameters. If the original non-linear equation is linearized about the origin ($\mathbf{A} = vI, \mathbf{t} = 0$) only small affine transforms can be solved for. By coarsely sampling the space of affine parameters and linearizing the non-linear equations at these sample values, larger affine transforms can be solved for. Only a coarse sampling is required for linearization reduces the number of samples required to solve for the affine transformation.

In the case of an affine transform, a linear approximation to $F_2$ is given by

$$F_1(\mathbf{r}) = F_2(\mathbf{A}\mathbf{r} + \mathbf{t}) \approx F_2(\mathbf{r}) + (\mathbf{t} + \mathbf{A}\mathbf{r})^T F_2'(\mathbf{r}) \tag{2.25}$$

This is a linear equation in the six unknowns $\mathbf{A}, \mathbf{t}$. An overconstrained system is obtained by assuming that the deformation over an entire region R is given by a single affine trans-

formation $\mathbf{A}, \mathbf{t}$. An error measure is constructed by taking the euclidean distance

$$\sum [F_1(\mathbf{r}_i) - F_2(\mathbf{A}\mathbf{r}_i)]^2 = \sum [F_1(\mathbf{r}_i) - F_2(\mathbf{r}_i) + (\mathbf{t} + \mathbf{A}\mathbf{r}_i)^T F_2'(\mathbf{r}_i)]^2 \tag{2.26}$$

where i indexes over the pixels in the region R. Since this is quadratic in the unknowns $\mathbf{A}, \mathbf{t}$, minimizing this with respect to the unknowns gives a set of linear equations in the unknowns.

A number of people have used the above approach. If there is assumed to be no deformation, this reduces to Lucas and Kanade's technique [] for finding flow. If the deformation is large, one patch will be significantly larger than the other. However, the formulation above assumes that patches of the same size are being matched which will, therefore, lead to poor flow estimates. Even if one wants to compute flow, better flow estimates are obtained by solving for the complete affine transform [11]. The disadvantage of computing the complete affine transform compared to just the flow (i.e. translation) is that since many more unknowns need to be computed, the regions R required to compute a robust solution must be larger.

Minimizing the above error measure with respect to the affine parameters gives an overconstrained linear system in the affine parameters. The affine parameters obtained are, usually, poor. [11]. Considerable improvement can be obtained by using a warp and iterate strategy: an initial estimate of the affine transform is computed, then one image is warped toward the other using this estimate and the procedure repeated for the warped image [6, 64]. This may be viewed as a minimization of the original error measure using a Gauss–Newton technique. Impressive results can be obtained if this is combined with a coarse-to-fine pyramid scheme [6]. The coarse-to-fine pyramid (see 2.4), essentially, provides a good way of computing large translations. Still, the technique is limited to solving for moderately sized affine deformations For large deformations the corresponding regions R differ greatly in size and shape. Since linearized techniques assume that the regions are approximately similar in shape and size, the method breaks down. On the other hand,

the approach used in this dissertation better approximates the region sizes so as to obtain improved results.

### 2.3.3 Deformation Based Methods

The techniques described below (except for Kass's) recover the affine transform by measuring the deformation of a region. If the affine transform is small, linearization can be done. Large affine transforms can be recovered by moving the linearization point. That is, the space of affine transforms can be coarsely sampled and linearization done about each sample.

Kass [31] convolved a pair of stereo images with the Gaussian and its first and second derivatives at a number of scales and used the best match over a set of filters to find correspondence. Although his technique only measures disparity, it is important in being one of the first to account for the effects of geometric distortion.

Consider a 1-D affine transform (i.e. a scale change and a translation). Given that $G_i$ denotes the $i^{th}$ derivative, Kass assumed that a small value of the following error function indicates a correspondence:

$$\sum_x |F_1(x) * G_i(x,\sigma) - F_2(x) * G_i(x,\sigma)|^2 \tag{2.27}$$

However, a small value of the error only indicates correspondence if the geometric deformation (i.e. the affine transform between the two patches) is small. Assuming that the two patches are related by an affine transform, the error

$$E_1 = \sum_x |F_1(x) * G_0(x,\sigma) - F_2(x) * G_0(x,\sigma)|^2 \tag{2.28}$$

due to geometric deformation may be approximated to first order by the quantity

$$E_1 = sF_2(x) * d^2G(x,\sigma)/dx^2 \tag{2.29}$$

where s is the scale change and $d^2G(x,\sigma)/dx^2$ is the second derivative of the Gaussian. Kass, therefore, used the result of filtering the image with the second derivative Gaussian as an indicator of the confidence in the correspondence. Wherever this was high, the confidence in the correspondence was low and vice-versa.

Werkhoven and Koenderink [75, 76] went further and tried to estimate the affine transform by matching representations of the two image patches. The representations were obtained by filtering one image at a point with Gaussian derivatives and the second with *deformed* versions of Gaussian derivatives, the deformation being precisely the affine transformation of the patch. The solution, therefore, consists of finding the appropriate deformed Gaussian such that the filter outputs from the two image patches are equal. For example, consider two images $F_1(\mathbf{r})$ and $F_2(\mathbf{Ar}+\mathbf{t})$ where the second image is an affine transformed version of the first.

$$F_1(\mathbf{r}) = F_2(\mathbf{Ar}+\mathbf{t}) \tag{2.30}$$

If the first image is filtered with a Gaussian $G(\mathbf{r},\sigma)$, then the second image must be filtered with a deformed Gaussian leading to the following equality (see Chapter 4 for a derivation).

$$F_1(\mathbf{r}) * G(\mathbf{r},\sigma) = F_2(\mathbf{r}_1) * G(\mathbf{A}^{-1}(\mathbf{r}_1+\mathbf{t},\sigma) det(\mathbf{A}^{-1}) \tag{2.31}$$

where $\mathbf{r}_1 = \mathbf{Ar}$. To avoid search, they linearized the filter outputs with respect to the affine parameters. For example, the Gaussian on the right hand side in equation (2.31) may be linearized with respect to the affine parameters (see Chapter 4) to give the following equation:

$$F_1 * G(.,\sigma) \approx F_2 * G(.,\sigma) + \sigma^2[(a_{11}-1)F_2 * G_{xx}(.,\sigma) + a_{12}F_2 * G_{xy}$$

$$+ \quad a_{21} F_2 * G_{xy}(., \sigma) + (a_{22} - 1) F_2 * G_{yy}] + \mathbf{t} \cdot [F_2 * G'(., \sigma) \quad (2.32)$$

where $\mathbf{r} = (x, y)$, the $a_{ij}$ are entries of the matrix $\mathbf{A}$, G' denotes the vector first derivative of a Gaussian while $G_{xx}, G_{xy}$ and $G_{yy}$ denote the three second derivatives of a Gaussian. This equation is linear in the unknown afffine parameters $\mathbf{A}, \mathbf{t}$. Werkhoven and Koenderink also linearized the first and second derivatives of a Gaussian in a similar fashion to give a total of six equations in six uknowns. The solution of the resulting linear system gave the six affine parameters.

Although this system is not underconstrained (there are six equations and six unknowns) the results obtained are poor [76] (see also Chapter 4). For even moderate deformations ($\leq 1\%$), the errors are of the order of 3 to 6% because the linearized equations are a poor approximation to the original non-linear equations. We show later (in Chapter 5) that the results can be improved by solving the original non-linear equations using a Gauss Newton approach. At each iteration of the Gauss–Newton technique, a system of equations of the above form is solved and the image warped according to the recovered affine transform. The use of spatial filters at multiple scales also improves the results (this will be discussed in Chapter 5). A second problem with the Werkhoven and Koenderink algorithm is that all the filtering is done at a single point. In this study (see Chapter 5), by pooling the filtered outputs from different points, robust solutions to the problem of matching two image patches under affine transforms are obtained.

### 2.3.4 Frequency Based Methods

Instead of using the spatial domain, the same computations can be carried out in the frequency domain. The frequency domain constraints can be obtained by taking the Fourier transform the original image equation $F_1(\mathbf{r}) = F_2(\mathbf{A}\mathbf{r} + \mathbf{t})$.

$$S_1(f) = S_2(\mathbf{A}^{-1} f) exp(-2\pi j f t) det A^{-1} \quad (2.33)$$

To compute the affine parameters between two patches requires that the Fourier transform be computed over those "local" patches rather than over entire image. Instead of explicitly computing a local Fourier transform, the spectrogram may be sampled using a set of filters (e.g. Gabor filters). A spectrogram of a window w(**r**) within an image F(**r**) is given by

$$S(\mathbf{r}, \mathbf{f}) = FoF(\mathbf{r}')w(\mathbf{r} - \mathbf{r}') \tag{2.34}$$

where $Fo$ is the Fourier transform and **r**' is a dummy variable. It can be shown that a sampled version of the spectogram can be computed using a set of Gabor filters [17]. The method is then equivalent to a filter based approach.

The translation between corresponding image patches can be recovered by comparing the phases of the Fourier transform of the image patches [74]. One advantage of the technique is that to some extent the translation and the deformation parameters can be decoupled by computing the square root of the power spectral density

$$|S_1(f)| = |S_2(\mathbf{A}^{-1}f)|detA^{-1} \tag{2.35}$$

Note that the translation no longer appears in this equation. In many situations this is advantageous, since this means that the method is insensitive to small translations or to small errors in translations.

One approach to solving this equation for the affine parameters was proposed by Malik and Rosenholtz [39]. To simplify the equations the determinant term was first eliminated by dividing by the average power spectral density and the equation was then linearized. This is equivalent to a frequency domain version of equation [2.25]. The affine transform was then computed by solving a linear set of equations. One of the main disadvantages of the method is that the frequency resolution of the Fourier transform is poor if the region over which the transform is computed is small (the frequency resolution is of the order of 1/N where N is a linear dimension of the image patch [53]). Thus, for good accuracy,

the window over which the Fourier transform must be computed must be large. These are much larger than the corresponding regions required for the spatial domain.

Instead of explicitly computing the power spectrogram, one can use many narrowly tuned filters to obtain an equivalent representation of the power spectrogram in the spatial domain. This has been done mainly for the case of optical flow [78] and for the special case of stereo [78].

### 2.3.5 Other Features or Basis Functions

A choice of appropriate features often simplifies the problem of computing affine transforms or finding the best transform from a subgroup of the affine transform. For completeness, a few of these techniques will be briefly mentioned.

For example, using a finite number of appropriate filters, the rotation subgroup may be recovered [55]. Such filters include the first and second derivatives of Gaussians. Under some conditions, a scale change can be recovered by using the Mellin Fourier transform [62]. The intuitive basis for this technique will now be explained. A scale change transforms the coordinates $\mathbf{r}_1$ in the first image to $\mathbf{r}_2 = s\mathbf{r}_1$. By transforming the original image using the natural logarithm, the new coordinates $\mathbf{x}_1$ and $\mathbf{x}_2$ are:

$$\mathbf{x}_2 = \mathbf{x}_1 + ln(s) \tag{2.36}$$

The coordinate transformation between $\mathbf{x}_1$ and $\mathbf{x}_2$ due to the scale change s is now a pure translation which can be recovered using standard techniques for recovering translation. Segman et al [62] show that if a coordinate transformation can be mapped to a Lie group and the generators of this group are Abelian, then the images can be mapped to a new set of features which differ by a translation. A similarity transform (scale, rotation and translation) is one such group. The general affine transform is not an Abelian group and, therefore, cannot be solved in this manner. The technique assumes that the entire image is transformed in the same manner. Thus this technique is not general enough for the

problems considered in this study which involve matching and recovering general affine transforms over a part of an image.

Other choices of filters or basis functions may be made to recover the affine transform. For example, Szeliski et al [69] used cubic splines to recover affine transforms. They represented the whole image in terms of finite elements. Locally each finite element was modelled using cubic splines. The transformation required to bring the two grids of finite elements into aligment is the transformation needed to match the two images.

## 2.4 Coarse-to-Fine Approach: Recovering Large Translations

Sometimes, the translation component of the affine transform may be large. A standard way to recover large translation components is to use a coarse–to–fine strategy [3, 6]. The coarse–to–fine strategy is to initially start with the low frequency components of the image and compute an initial estimate of the translation. The estimate may then be refined by using higher frequency components. The strategy does not require explicit computation of the Fourier transform. Instead, the image is succesively smoothed at different scales. The smoothing filter usually used is a Gaussian.

The rationale for the coarse-to-fine strategy can be understood by considering a Fourier decomposition of an image. For a given frequency $\mathbf{f}$ and a given translation $\mathbf{t}$, the only change in the Fourier component is a phase shift by the angle $\mathbf{f} \cdot \mathbf{t}$. The phase shift can be made arbitrarily small by picking a sufficiently small frequency $\mathbf{f}$. Thus, large translations can be recovered at coarse scales since the resultant phase shift is small.

Directly smoothing with large Gaussian masks is expensive. However, the same effect can be obtained by constructing a Gaussian pyramid. This is done as follows. The first level of the pyramid is the image itself. The next level is obtained by filtering the image with a small Gaussian mask and sub-sampling by a factor of 2. Further pyramid levels are obtained by recursively following the same procedure i.e. filtering with a Gaussian with

the same standard deviation and sub-sampling by a factor of 2. A similar procedure may also be used to create pyramids using other Gaussian derivatives.

The technique is valuable when used to match entire images with each other. However, it may not be applicable to situations where the image patch to be matched may only be a small portion of the entire image. The technique cannot also be generalized to the non-translation component of the affine transform. This is because the deformations cause both the magnitude, the phase and the actual frequencies to change.

## 2.5 Recovering Deformations in the Context of Shape from Texture

There are two aspects to finding shape from texture. First, image measurements need to be made. Second, the image measurements are used with constraints on the texture model and a projection model to derive 3-D information. The focus here will be on the image measurements rather than on deriving the 3-D shape. However, since the image measurements required are determined by the texture model, a discussion of such models is necessary for a complete treatment of the subject.

A number of methods to recover shape from texture are based on recovering image deformations. The deformation may be recovered by comparing two image patches. More commonly assumptions are made on the distribution of the texture. Any deviation from the assumed distribution is assumed to be due to the non-frontal projection of the surface. The measured deviation may, therefore, be used to reconstruct the local surface orientation. The assumed distribution simplifies the problem so that the complete affine transform does not need to be recovered, only subgroups of the affine deformation are measured. A discussion of methods to recover affine transforms for the shape from texture problem is, therefore, incomplete without a treatment of the assumed distributions. Some common distributions that are assumed include [65]

1. Homogeneity: Assumes that the texture density is approximately constant. Any variation in the density (area) of the images of the texture elements is, therefore, due to

either an increasing or decreasing distance from the camera or due to foreshortening. Foreshortening can occur with either orthographic or perspective projection while a change in density with distance occurs solely due to perspective projection. A large class of methods have attempted to find shape from texture by measuring the area or density gradient [2, 26, 66, 48]. Since by assumption the texture density is the same everywhere, the correspondence does not need to be recovered.

2. Isotropy: Assumes that the measured property is isotropically distributed. Two different isotropy measures have been used in the literature. The first uses local edge directions. In strong isotropy [77], the edges are assumed to be isotropically distributed whereas weak isotropy [17] only requires that the edges have no preferential direction on the average. The second measure of isotropy used assumes that the second moment matrix of the power spectral density of the surface texture is given by some constant times the identity matrix. The actual measured second moment matrix, therefore, encodes the distortion caused by the slant and tilt of the texture patch [9, 17, 36]. Note that no correspondence problem exists since what is being measured is the deformation with respect to an assumed distribution rather than the deformation between two image patches.

3. Homotropy: The assumption here is that on the average there is a preferred shape and orientation to the tangent distributions. For example, if the texture elements are all ellipses, homotropy would imply that the ellipses have the same shape and they are oriented in the same direction on the surface. Homotropy is valid only for planes since a constant direction cannot be defined for curved surfaces. A weaker version of homotropy assumes that any variation in orientation or shape is smooth and slowly varying [65]. The spacing between the texture elements may be random.

### 2.5.1 Shape Distortion

An alternative technique for computing shape from texture makes no assumption on the distribution of the texture. Rather it is assumed that the texels are identical and that there is some subgroup of an affine distortion between the two image texel patches. It may be shown that this is an underconstrained problem if two texels are used [30]. A common solution is to assume that the surface orientation of one of the elements is known [30]. Alternatively, a comparison of the projected areas of texels may also be used to recover the surface orientation of planes [5, 52, 7]. Explicit correspondence between texel elements is usually required. Often this technique is used to compare areas of corresponding texels.

Real world textures may satisfy one or more of these assumptions at the same time. Early techniques to recover shape from texture concentrated on recovering surface orientation and simplified the problem of making image measurements. Often very specific assumptions on the texture patterns were made. For example, the image was sometimes assumed to consist of black dots against a white background and texels were recovered by thresholding [2]. Recent shape from texture techniques have tried to deal with more general textures. Since the focus of this work is on recovering image deformations, our discussion will now focus on methods which compute deformations either between two image patches or between an image patch and an ideal patch (for a review of other techniques to compute shape from texture see [65]). Essentially such methods use moments of some function of the power spectral density or spatial filters. There is a close relationship between filter based methods and moment techniques. For example, Gaussian and Gaussian derivative filters can always be expressed as a sum of Gaussian weighted moments. Moments computed over image patches are related by an affine transform.

### 2.5.2 Moment Based Methods

Under orthographic projection a planar textured surface patch $V(\mathbf{r})$ is related to its image projection $F(\mathbf{r})$ by an affine deformation.

$$V(\mathbf{r}) = F(\mathbf{A}\mathbf{r}) \tag{2.37}$$

The affine deformation is related to the slant $\sigma$ and tilt $\tau$ angles by [9]

$$\mathbf{A} = \mathbf{R}(\tau)\mathbf{M}(\sigma)\mathbf{R}(-\tau) \tag{2.38}$$

where $\mathbf{R}$ is a 2-D rotation matrix, and the slant matrix is

$$M(\sigma) = \begin{bmatrix} \cos\sigma & 0 \\ 0 & 1 \end{bmatrix} \tag{2.39}$$

Given the affine deformation the slant and tilt angles can, therefore, be recovered up to a $\pm 180\,\mathrm{deg}$ ambiguity in the tilt.

Fourier transforming equation 2.37 gives

$$S_v(\mathbf{f}) = det(\mathbf{A}^{-1})S(\mathbf{A}^{-1}\mathbf{f}) \tag{2.40}$$

where $S_v$ and S denote the Fourier transforms of V and F respectively. The second moments of the magnitude of the Fourier response are therefore related by [17]

$$\int \mathbf{f}\mathbf{f}^T|S_v(\mathbf{f})|d\mathbf{f} = \int \mathbf{A}\mathbf{A}^{-1}\mathbf{f}(\mathbf{A}^{-1}\mathbf{f})^T|S(\mathbf{A}^{-1}\mathbf{f})|(\mathbf{A}^T)d\mathbf{A}^{-1}\mathbf{f} \tag{2.41}$$

This may be rewritten as

$$\mu_s = \mathbf{A}\mu\mathbf{A}^T \tag{2.42}$$

where $\mu_s$ and $\mu$ denote the second moment matrices of $S_v$ and $S$ respectively.

Given $\mu_s$, the affine transform can be recovered. In general the form of $\mu_s$ is unknown and all one can do is to make either statistical assumptions on its form (isotropy and homogeneity) or assume that there are identical texels distributed over the image.

#### 2.5.2.1 Isotropy

One possible isotropic assumption [9, 17] is that $\mu_s$, the second moment matrix of the surface texture, is equal to the identity matrix. i.e.

$$\mu_s = \mathbf{I} \tag{2.43}$$

Then

$$\mu = \mathbf{A}^2 \tag{2.44}$$

Intuitively, the above equation is consistent with the notion that an affine transform maps a circle into an ellipse. This follows from the fact that

$$x^T \mu_s x = x^T x = c \tag{2.45}$$

is the equation of a circle, while

$$x^T \mu x = x^T A^T A x = c' \tag{2.46}$$

represents the equation of an ellipse. The slant and tilt of a planar surface can now be recovered using the eigenvalues of $\mu$.

Brown and Shyvaster [9] computed the orientation of a global plane using isotropy. Instead of using spectral moments they used the auto-correlation function of the intensity which turns out to be the Fourier transform of the power spectral density. These equations can also be used for weak perspective projection by incorporating an additional scale term $s$ in equation 2.38 to give

$$\mathbf{A} = s\mathbf{R}(\tau)\mathbf{M}(\sigma)\mathbf{R}(-\tau) \tag{2.47}$$

By comparing different patches on the surface, the relative scale may be determined from the smaller eigenvalue of $\mu$ assuming that there are no lighting or shading variations.

For perspective projection with a planar retina, this equation is only valid near the image center, since at other points the image plane is no longer normal to the view ray and hence the projection changes with image position. Garding [17] was able to circumvent this problem using a spherical retina. A spherical retina has the property that locally it is always normal to the view ray. Hence the equations for weak perspective can be used at every point. A virtual spherical retina can be created using a gaze transformation. This transformation is only dependent on camera parameters. Garding computed the required moments using Gabor filters. The basic technique assumes that a single plane is imaged. Garding partitioned the image into a number of windows and computed moments over each such window. It was assumed that within a window, the surface could be represented by a plane. Thus a coarse piecewise planar approximation to the surface was computed.

This model has a number of advantages for computing shape from texture. Since the comparison is between an image patch and an ideal model of the surface texture rather than between two image patches, no explicit correspondence needs to be computed. The algorithm uses the shape of the ellipse which is computed using ratios of moments. For the case of orthographic projection the absolute values of the moments are, therefore, inconsequential. This makes the method relatively insensitive to lighting. The main disadvantage of the technique is that although the method is reasonably robust to some deviations from isotropy, the assumption of isotropy is not always valid for real textures.

#### 2.5.2.2 Homogeneity

An alternative assumption to make is to assume that the surface texture is homogeneous. This implies that the local second (surface) moment matrix is more or less constant at neighboring points (it is implicitly assumed that locally the surface can be approximated by a plane and the moments are computed over this plane). If the surface orientation at any point on the surface is known, then it can be used to recover the surface orientation of every other point. Taking the determinant of both sides of equation 2.42 gives

$$det(A)^2 det(\mu_s) = cos(\sigma)^2 det(\mu_s) = det(\mu) \tag{2.48}$$

If now two points in the image are taken and their surface moments are assumed to be the same, then

$$cos(\sigma_1)^2 det(\mu_2) = cos(\sigma_2)^2 det(\mu_1) \tag{2.49}$$

The computation of slant depends only on the ratio of the determinants of moments i.e. it depends on the shapes of the ellipses defined by the second moments rather than the absolute magnitudes of the moments. This should make it relatively insensitive to lighting and albdeo variations.

The limits for the original moment equations were $-\infty$ to $\infty$. However, the approach discussed here requires the use of local moments computed by using finite windows. If finite windows are used, the windows must be different for the two corresponding image patches. To see this note that a circular window on the surface corresponds to an elliptical window in the image. By homogeneity, circular patches of the same size on the surface have on average the same energy and second moments. Therefore corresponding elliptical patches (and these ellipses may have different shapes and size) must also have the same energy. To compare them, one must therefore compute moments over the appropriate elliptical areas. Since the size and shape of the windows depends on the affine deformation between the patches - which is unknown - the use of different windows would make the equations complicated and messy. It is, therefore, usually assumed that in practice using the same window suffices. Recently some techniques have been suggested for computing the window adaptively (see Section 2.5.2.3).

It is often desirable to make the technique relatively insensitive to lighting and albedo variations. This can be done by dividing by the local energy, which is just the power spectral density integrated over the window. The local energy in the neighborhood of each image point is related to the surface energy of the corresponding point by

$$\int |S_v(\mathbf{f})|^2 df = \int |S(\mathbf{Af})|^2 detAd\mathbf{Af} \tag{2.50}$$

Dividing the second moment equation 2.42 by the above equation gives

$$A\mu_s{}'A^T det(A) = \mu' \tag{2.51}$$

where the prime denotes normalized moments. Thus the determinants are now related by

$$det(\mathbf{A})^4 det(\mu_s{}') = \cos(\sigma)^4 det(\mu') = det(\mu') \tag{2.52}$$

The slants at neighboring points are, therefore, related by

$$\cos(\sigma_1)^4 det(\mu_2{}') = \cos(\sigma_2)^4 det(\mu_2{}') \tag{2.53}$$

Notice that the ratio of the determinants of the second moments are now proportional to the fourth power of the cosines of the slants unlike the un-normalized equations where they were only related by their second powers.

Super and Bovik [67] used the normalized equations to recover shape from texture. They assumed that at one point, the surface was frontal (i.e. its slant was zero). Knowing this the surface orientation at every other point can be recovered and they were able to recover the surface orientations for curved objects.

#### 2.5.2.3 Adaptive Filtering

As discussed in the previous section, when finite windows are used, a given window on a surface patch will correspond to windows of different sizes and shapes in the image depending on viewpoint. Since the window shape and size depend on the unknown affine deformation between the patches, the correct window with which to filter is uknown apriori. However, a couple of adaptive schemes to compute the windows have been suggested. Both

techniques use Gaussian and Gaussian derivative filters. The idea is simply to first compute an approximate value of the affine transform using a circular Gaussian. Then the image is locally re-filtered by an elliptical Gaussian whose parameters are given by the current estimate of the affine transform and the affine transform recomputed. The process can be repeated till convergence is achieved; let

$$G(\mathbf{r}, \sigma) = k_1 exp(-\mathbf{r}^T \mathbf{r} \sigma^2) \tag{2.54}$$

where $k_1$ is a normalization factor define a circular Gaussian. Then the elliptical Gaussian is defined by

$$G(\mathbf{r}, \sigma) = k_2 exp(-\mathbf{r}^T \mathbf{D}^{-1} \mathbf{r}) \tag{2.55}$$

where $k_2$ is a normalization factor, $\mathbf{D} = \mathbf{A'}\mathbf{A'}^{-1}$ is a real symmetric matrix and $\mathbf{A'}$ is the current estimate of the affine matrix.

Warping is an alternative adaptation scheme that is commonly used in the motion and registration literature. Many of the common schemes use filter outputs at many corresponding points to compute the affine transforms An affine transform causes the relative spacing between such points to change. An adaptive scheme makes it necessary to use some kind of interpolation to compare the filter outputs at corresponding points in the two images. This can be achieved by warping. However, in most shape from texture work, the affine transform at a point is usually computed using only filter outputs/moments at that point and hence warping is not necessary.

#### 2.5.2.4 Picking Points

So far it has been assumed that textural information is available everywhere in the image. This ia obviously not true for many non-homogeneous textures. For such textures, the information at some points may be more reliable than at others. Lindeberg and Garding [36] considered the problem of finding reliable feature descriptors. The descriptor they

used is the following second moment computed as the average of the outer product of the first derivative of the Gaussian.

$$\mu_l = \int w(x,y)(F * G')(F * G')^T dxdy \tag{2.56}$$

where G' is the first derivative of the Gaussian, F is the image irradiance, w is a Gaussian window function whose standard deviation is termed the integration scale. They show that this is in fact equivalent to using Gaussian weighted second moments of the power spectral density.

Lindeberg and Garding were in addition interested in finding the appropriate scale (integration scale) over which to compute the above descriptor. They argued that the maxima and minima of some function of the second moment matrix $\mu_l$ over scale and position were good candidates for interesting features. They chose the local maxima of the $det(\mu_l)$ over scale and position. They then used both isotropy and the areas of detected blobs to compute planar orientations.

### 2.5.3 Comments

The shape from texture methods discussed here have largely been limited to reconstructing globally planar surfaces or coarse piecewise planar surfaces. Of the moment based techniques, only Super and Bovik [67] attempt to reconstruct a curved surface. However, no information is available on the quality of their reconstructions. In addition, no technique attempts to solve the problem of recovering shape for a set of identical texels distributed arbitrarily on a surface. In principle this could be solved by computing the affine transform (shape distortion) between corresponding pixels. However, the difficult problem of finding correspondences between texels then needs to be solved; this is particularly difficult because the distances between texels (texels is used in a larger sense here) may be large. Our discussion, here, has not included the work of Malik and Rosenholz [39]. This technique is

similar to work in the area of motion and registration and was more appropriately discussed there (see Section 2.3.4).

Shape from texture techniques are usually insensitive to surfaces which are nearly frontal. This is reflected by the fact that all shape from texture methods compute the cosine of the slant matrix in order to compute the slant. Since the cosine function varies slowly near zero degrees, these techniques are not very sensitive to surfaces which are nearly frontal ($\cos(0) = 1.0, \cos(10) = 0.98$).

# CHAPTER 3

# PROBLEM FORMULATION

## 3.1 Introduction

The problem of matching two image patches under an affine transformation is formulated here as a problem of matching appropriate representations of the patches with respect to each other. The representations are obtained by spatially filtering the image patches with Gaussians and derivatives of Gaussians. It is shown that by using Gaussian filters, the outputs are equal provided the Gaussian is affine-transformed in the same manner as the function. Similar results are derived for the first and second derivatives of Gaussians. The problem of recovering the affine parameters is then reduced to finding the deformation of a known function, the Gaussian, rather than of the unknown brightness functions.

The basic intuition behind these statements is illustrated by the one-dimensional (1D) functions plotted in Figures 3.1 and 3.2. Figure 3.1 shows a 1D function and a Gaussian filter with scale $\sigma = 1$. In Figure 3.2 the 1D function is scaled by a factor of 1.4 and the Gaussian's standard deviation is scaled by the same amount. Note that at corresponding points in the two graphs, the product of the function and the Gaussian remains the same. Thus, the filter outputs remain the same (the Gaussians must be normalized to have unit area for the filter outputs to be the same).

The solution to the problem consists of finding the appropriate deformed Gaussian(s) such that the filter outputs from the two image patches are equal. This can be done by sampling the space of affine transformations, which is an expensive operation. Alternatively, the space of affine transformations can be coarsely sampled and interpolation used to solve

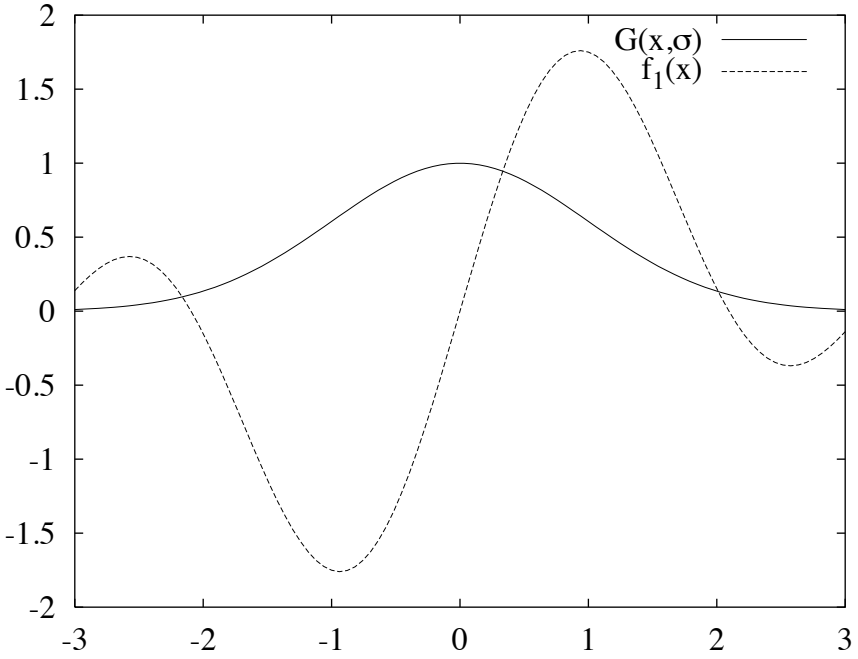



**Figure 3.1.** A 1D function and a Gaussian filter overlaid on it.

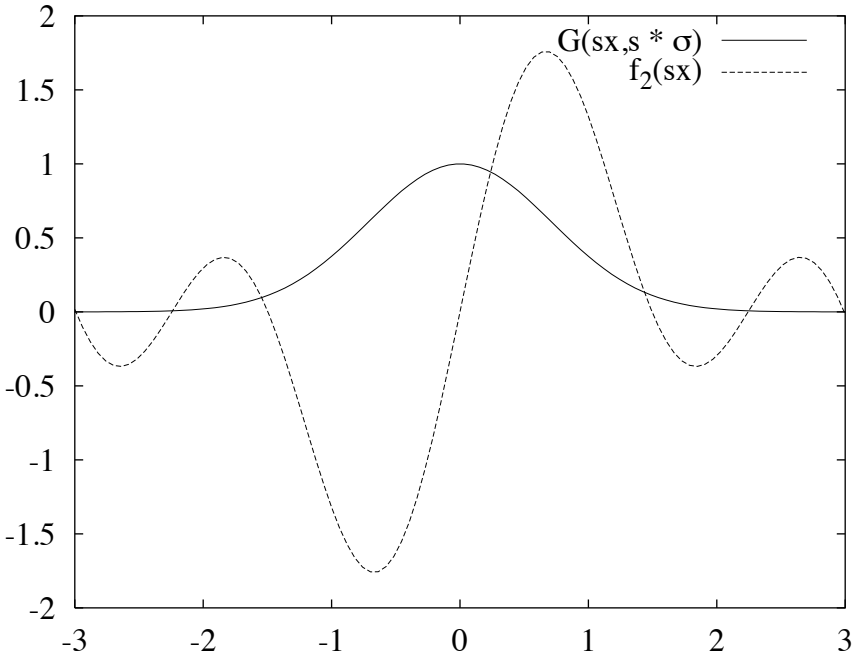



**Figure 3.2.** A scaled version of the 1D function and a scaled Gaussian filter overlaid on it. Note that the Gaussian and the 1D function are scaled by the same amount.

for affine transforms which lie in between the sampled points. This is the procedure that will be followed here (see Chapter 4 and Chapter 5).

In addition by pooling the filter outputs from a number of different points over a patch, the results can be made much more robust. This contrasts with other work which also models deformations [75] but is much more brittle because it uses filter outputs from a single point.

This chapter is laid out as follows. Given two image patches which differ by an affine transformation, it is shown that the output obtained by filtering the first image patch with a Gaussian must be equal to the result of filtering the second image patch with a deformed Gaussian, the deformation being precisely the affine transform. Similar results are then derived for the first and second derivatives of Gaussians. The above equations relating the filtered outputs of the image patches are derived by filtering with the Gaussians and the derivatives centered at the origin. The result of filtering with Gaussians and their derivatives at other points is then derived. By taking the Fourier transform of the Gaussian equations, similar relationships are derived in the frequency domain. The initial discussion assumes that the image translation is zero, which is equivalent to assuming that the image translation is known. Methods for incorporating the image translation into the formulation are then discussed.

It is assumed that shading and illumination effects can be ignored. These can, however, be taken care of by multiplying the equations by an additional constant factor. Most of what follows is valid for arbitrary dimensions; the 2-D case is used as a specific example.

## 3.2 Deformation of Filters

**Notation** Vectors will be represented by lowercase letters in boldface, while matrices will be represented by uppercase letters in boldface.

Consider two functions $F_1$ and $F_2$ related by an affine transform $(\mathbf{A}, \mathbf{t})$. For the time being, the translation will be assumed to be zero. The effects of translation will be discussed later in Section 3.4.

Since the affine transform may be viewed as a transformation of the underlying coordinate system, this may be written as

$$F_1(\mathbf{r}) = F_2(\mathbf{Ar}) \tag{3.1}$$

The areas under the functions over some finite interval are related by:

$$\int_{-\mathbf{a}}^{\mathbf{a}} F_1(\mathbf{r})d\mathbf{r} = \int_{-\mathbf{a}}^{\mathbf{a}} F_2(\mathbf{Ar})d\mathbf{r} = \int_{-\mathbf{Aa}}^{\mathbf{Aa}} F_2(\mathbf{Ar})d(\mathbf{Ar}) \times \det \mathbf{A}^{-1} \tag{3.2}$$

expressed succinctly as

$$\nu_1 = \nu_2 \times \det \mathbf{A}^{-1} \tag{3.3}$$

where $\nu_1 = \int_{-\mathbf{a}}^{\mathbf{a}} F_1(\mathbf{r})d\mathbf{r}$ and $\nu_2 = \int_{-\mathbf{Aa}}^{\mathbf{Aa}} F_2(\mathbf{Ar})d(\mathbf{Ar})$. Let $s_i$ be the scale change along the ith dimension and n the number of dimensions. The fact that $\det(\mathbf{A}) = \Pi_1^n s_i$, the product of the scale changes $s_i$, leads to an intuitive explanation for equation (3.3). Consider the 1-D case (n = 1), where the affine transform reduces to a scale change. Assume that the function $F_1$ is graphed on a rubber sheet. The graph of $F_2$ is obtained by stretching the sheet (assuming the sheet stretches linearly). Along with the function, the coordinate axes and the units marked off along them are also stretched. The determinant term is equal to the stretching undergone by the coordinates. Note that the area under a function may also be viewed as the zeroth moment of the function.

Equation (3.2) cannot be used directly because the limits on the right-hand side depend on the affine transform and are therefore unknown. This crucial point has not been handled correctly before. Instead, a number of existing methods [6, 11] assume that the patches are the same size (i.e. the limits are the same). One approach would be to take the limits

from $-\infty$ to $\infty$. However, this will not preserve localization. Localization is desirable because it is often necessary to compute affine transforms between image patches rather than between entire images. The solution described here involves weighting the function by another function which decays rapidly - here the Gaussian will be used. Note that filtering with a Gaussian or a Gaussian derivative may be viewed as the same as computing a linear combination of Gaussian weighted moments. Thus a filtering operation with a Gaussian or a derivative of a Gaussian will sometimes be referred to as a weighted moment in the rest of the dissertation. Weighted moments differ from ordinary moments in one important respect; invariants to the general affine transform cannot be constructed using weighted moments.

The case where $\mathbf{A} = s\mathbf{R}$, i.e. the affine transform is composed of a scale change s and a rotation $\mathbf{R}$, will be discussed first; this is followed by a discussion of the more general case.

### 3.2.1 Case $\mathbf{A} = s\mathbf{R}$

Denote the un-normalized Gaussian by

$$H(\mathbf{r}, \sigma^2) = exp(-\mathbf{r}^T\mathbf{r}/2\sigma^2) \tag{3.4}$$

Multiply both sides of equation (3.1) by $H(\mathbf{r}, \sigma^2)$ to obtain :

$$F_1(\mathbf{r})H(\mathbf{r}, \sigma^2) = F_2(\mathbf{Ar})H(\mathbf{r}, \sigma^2) \tag{3.5}$$

The following identity follows from the orthonormality of rotation matrices

$$H(\mathbf{r}, \sigma^2) = \exp(-\mathbf{r}^T\mathbf{r}/2\sigma^2) \;\; = \;\; \exp(-s\mathbf{r}^T R^T Rs\mathbf{r}/2(s\sigma)^2)$$

$$= \quad H(s\mathbf{Rr}, (s\sigma)^2) = H(\mathbf{Ar}, (s\sigma)^2) \tag{3.6}$$

and allows equation (3.5) to be rewritten as

$$F_1(\mathbf{r})H(\mathbf{r}, \sigma^2) = F_2(\mathbf{Ar})H(\mathbf{Ar}, (s\sigma)^2). \tag{3.7}$$

The Gaussian weighted zeroth moment (which is identical to filtering with a Gaussian) may, therefore, be written as

$$\begin{aligned} \int F_1(\mathbf{r})H(\mathbf{r}, \sigma^2)d\mathbf{r} &= \int F_2(\mathbf{Ar})H(\mathbf{Ar}, (s\sigma)^2)d\mathbf{r} \\ &= \int F_2(\mathbf{Ar})H(\mathbf{Ar}, (s\sigma)^2)d(s\mathbf{Rr})s^{-n} \end{aligned} \tag{3.8}$$

where the limits are taken from $-\infty$ to $\infty$. The factor $s^{-n} = \det \mathbf{A}^{-1}$ and can be eliminated by using normalized Gaussians

$$G(\mathbf{r}, \sigma^2) = \frac{1}{(2\pi)^{n/2}\sigma^n} H(\mathbf{r}, \sigma^2) \tag{3.9}$$

in place of H. The moment equation then becomes:

$$\begin{aligned} \int F_1(\mathbf{r})G(\mathbf{r}, \sigma^2)d\mathbf{r} &= \frac{1}{(2\pi)^{n/2}\sigma^n} \int F_1(\mathbf{r})H(\mathbf{r}, \sigma^2)d\mathbf{r} \\ &= \frac{1}{(2\pi)^{n/2}\sigma^n} \int F_2(\mathbf{Ar})H(\mathbf{Ar}, (s\sigma)^2)d(\mathbf{Ar})s^{-n} \\ &= \int F_2(\mathbf{Ar})G(\mathbf{Ar}, (s\sigma)^2)d(\mathbf{Ar}) \end{aligned} \tag{3.10}$$

The integral may be interpreted as the result of convolving the function with a Gaussian at the origin. It may also be interpreted as the result of a filtering operation with a Gaussian. To emphasize these similarities, equation (3.10) may be written as

$$F_1 * G(\mathbf{r}, \sigma^2) = F_2 * G(\mathbf{r}_1, (s\sigma)^2), \tag{3.11}$$

where $\mathbf{r}_1 = \mathbf{Ar}$.

Equation (3.11) is exact and is valid for arbitrary dimensions. The problem of recovering the affine parameters has been reduced to finding the deformation of a known function, the Gaussian, rather than of the unknown brightness functions. The basic intuition is that by using Gaussian filters, the outputs are equal provided the Gaussian is affine-transformed in the same manner as the function. The equation is invariant to rotation which implies that scale can be recovered but not rotation (the rotation can be recovered by other means to be discussed later). Note that although the limits are infinite, since the Gaussian is a rapidly decaying function, it suffices in practice to take limits from $-4\sigma$ to $4\sigma$ (and correspondingly from $-4s\sigma$ to $4s\sigma$ on the right hand side), maintaining spatial localization.

### 3.2.2 General Case

A similar equation can be shown to hold for arbitrary affine transforms, provided generalized Gaussians are used. Define a generalized Gaussian as

$$G(\mathbf{r}, \mathbf{M}) = \frac{1}{(2\pi)^{n/2}\det(\mathbf{M})^{1/2}} exp(-\frac{\mathbf{r}^T\mathbf{M}^{-1}\mathbf{r}}{2}) \tag{3.12}$$

where $\mathbf{M}$ is a symmetric positive semi-definite matrix. Then

$$\begin{aligned} G(\mathbf{r}, \sigma^2\mathbf{I}) &= \frac{1}{(2\pi)^{n/2}\det(\sigma^2\mathbf{I})^{1/2}} exp(\frac{-\mathbf{r}^T\mathbf{r}}{2\sigma^2}) \\ &= \frac{1}{(2\pi)^{n/2}\sigma^n} exp(\frac{-\mathbf{r}^T\mathbf{A}^T(\mathbf{A}^T)^{-1}\mathbf{A}^{-1}\mathbf{A}\mathbf{r}}{2\sigma^2}) \\ &= \frac{1}{(2\pi)^{n/2}\sigma^n} exp(-\frac{(\mathbf{A}\mathbf{r})^T(\mathbf{A}\mathbf{A}^T)^{-1}(\mathbf{A}\mathbf{r})}{2\sigma^2}) \\ &= \det(\mathbf{A}\mathbf{A}^T)^{1/2} G(\mathbf{A}\mathbf{r}, \sigma^2(\mathbf{A}\mathbf{A}^T)) \end{aligned} \tag{3.13}$$

The Gaussian weighted zeroth moment equation (i.e. the result of filtering with the Gaussian) may, therefore, be written

$$\int F_1(\mathbf{r})G(\mathbf{r}, \sigma^2 I)d\mathbf{r} = \det(\mathbf{A}) \int F_2(\mathbf{A}\mathbf{r})G(\mathbf{A}\mathbf{r}, \sigma^2(\mathbf{A}\mathbf{A}^T))d\mathbf{r}$$

$$= \det(\mathbf{A}) \int F_2(\mathbf{Ar}) G(\mathbf{Ar}, \sigma^2(\mathbf{AA}^T)) d(\mathbf{Ar}) \det(\mathbf{A}^{-1})$$
$$= \int F_2(\mathbf{Ar}) G(\mathbf{Ar}, \sigma^2(\mathbf{AA}^T)) d(\mathbf{Ar}), \quad (3.14)$$

where the identity $\det(\mathbf{AA}^T)^{1/2} = \det(\mathbf{A})$ has been used. The matrix $\mathbf{AA}^T$ is a symmetric, positive semi-definite matrix and may therefore be written

$$\sigma^2 \mathbf{AA}^T = \mathbf{R\Sigma R}^T \quad (3.15)$$

where $\mathbf{R}$ is a rotation matrix and $\mathbf{\Sigma}$ a diagonal matrix with entries $s_1\sigma^2, s_2\sigma^2 ... s_n\sigma^2$ ($s_i \geq 0$). Thus

$$\int F_1(\mathbf{r}) G(\mathbf{r}, \sigma^2 \mathbf{I}) d\mathbf{r} = \int F_2(\mathbf{Ar}) G(\mathbf{Ar}, \mathbf{R\Sigma R}^T) d(\mathbf{Ar}). \quad (3.16)$$

Again, to show the connections to convolution and filtering, this may be written as

$$F_1 * G(\mathbf{r}, \sigma^2 \mathbf{I}) = F_2 * G(\mathbf{r}_1, \mathbf{R\Sigma R}^T). \quad (3.17)$$

The level contours of the generalized Gaussian are ellipsoids rather than spheres. The tilt of the ellipsoid is given by the rotation matrix while its eccentricity is given by the matrix $\mathbf{\Sigma}$, which is actually a function of the scales along each dimension. The equation clearly shows that to recover affine transforms by filtering, one must deform the filter appropriately - a point ignored in most previous work [6, 12, 28]. This equation alone does not permit the recovery of the complete affine matrix - only the scales and the tilt. To find the complete affine transform, either higher order moments or derivatives of Gaussians (see Section 3.3) need to be considered and/or the filtering must be done at many points (see Section 3.5). Using higher order moments also permits the use of more overconstrained equations.

### 3.2.3 Using Other Weighting Functions

Weighting functions other than the Gaussian may be used. The weighting function must have certain desirable properties. It should be maximum at the origin and then decay, so that

points away from the origin are weighted less. Further, points equidistant from the origin should be weighted symmetrically. It should also be smooth. Differentiable functions of the form $f(\mathbf{r}/\sigma)$ are good candidates, although these are not rotation invariant. If rotation invariance is desired, functions which have the form $f(\mathbf{r}^t\mathbf{r}/\sigma^2)$ may be used. For example one possible weighting function is

$$\begin{aligned} w(\mathbf{r}) &= cos(\mathbf{r}^T\mathbf{r}/\sigma^2)\ for -\pi/2 \leq |\mathbf{r}/\sigma| \leq \pi/2 && (3.18) \\ &= 0\ otherwise && (3.19) \end{aligned}$$

Note that this is not normalized, and normalization is a desirable property because then weight functions at different scales can be directly compared. Other possibilities include Gabor functions. Weighting functions other than Gaussians or Gaussian derivatives will not be investigated further here.

## 3.3 Higher Order Moments and Derivatives of Gaussians

We will now derive equations relating the first moments of $F_1$ and $F_2$ under an affine transformation. Similar results will be derived relating the second moments of $F_1$ and $F_2$. Filtering with Gaussian derivative equations is equivalent to computing Gaussian weighted moments. Thus, similar relationships between the Gaussian derivatives of $F_1$ and $F_2$ will also be derived here.

The first order moments of $F_1$ and $F_2$ are related by

$$\begin{aligned} \int_{-\mathbf{a}}^{\mathbf{a}} \mathbf{r}F_1(\mathbf{r})d\mathbf{r} &= \int_{-\mathbf{a}}^{\mathbf{a}} \mathbf{r}F_2(\mathbf{Ar})d\mathbf{r} \\ &= \mathbf{A^{-1}} \int_{-\mathbf{Aa}}^{\mathbf{Aa}} \mathbf{Ar}F_2(\mathbf{Ar})d(\mathbf{Ar}) \det A^{-1} && (3.20) \end{aligned}$$

or

$$\mu_{\mathbf{1}} = \mathbf{A^{-1}}\mu_{\mathbf{2}} \times \det A^{-1} \qquad (3.21)$$

The second order moments are given by

$$\begin{aligned}\int_{-a}^{a} \mathbf{r}\mathbf{r}^T F_1(\mathbf{r}) d\mathbf{r} &= \int_{-\mathbf{a}}^{\mathbf{a}} \mathbf{r}\mathbf{r}^T F_2(\mathbf{A}\mathbf{r}) d\mathbf{r} \\ &= \mathbf{A^{-1}} \int_{-\mathbf{Aa}}^{\mathbf{Aa}} \mathbf{A}\mathbf{r}(\mathbf{A}\mathbf{r})^T F(\mathbf{A}\mathbf{r}) d(\mathbf{A}\mathbf{r})(\mathbf{A^{-1}})^T \det A^{-1} \end{aligned} \quad (3.22)$$

and this may be expressed as

$$\mathbf{\Gamma_1} = \mathbf{A^{-1}}\mathbf{\Gamma_2}(\mathbf{A^{-1}})^T \det(\mathbf{A^{-1}}) \quad (3.23)$$

Note that the zeroth moment equation is a scalar equation, the first moment a vector equation, and the second moment is a matrix equation. As before, the moments are not directly measurable in the form given and need to be weighted. If Gaussian weights are used, a simple derivation is available. This uses the fact that the derivatives of Gaussians are closely related to the weighted moments using Gaussians (Gaussian derivatives are linear combinations of products of polynomials with Gaussians).

The effect of filtering with derivatives of Gaussians (i.e. convolution with derivatives of Gaussians) can be obtained by differentiating the Gaussian equation (3.14)). Let us write $\mathbf{r}_1 = \mathbf{A}\mathbf{r}$ and then differentiate equation (3.14) as follows:

$$\begin{aligned} d[F_1(\mathbf{r}) * G(\mathbf{r}, \sigma^2\mathbf{I})]/dr &= d[F_2(\mathbf{r}_1) * G(\mathbf{r}_1, \mathbf{A}\mathbf{A}^T\sigma^2)]/d\mathbf{r} & (3.24) \\ &= d[F_2(\mathbf{r}_1) * G(\mathbf{r}_1, \mathbf{A}\mathbf{A}^T\sigma^2)]/d\mathbf{r}_1 \times d(\mathbf{A}\mathbf{r})/\mathbf{\dot{r}} & (3.25) \end{aligned}$$

This gives

$$F_1(\mathbf{r}) * dG(\mathbf{r}, \sigma^2\mathbf{I})/d\mathbf{r} = \mathbf{A}^T F_2(\mathbf{r}_1) * dG(\mathbf{r}_1, \mathbf{A}\mathbf{A}^T\sigma^2)/d\mathbf{r}_1 \quad (3.26)$$

where

$$dG(\mathbf{r}, \sigma^2\mathbf{I})/d\mathbf{r} = -\frac{\mathbf{r}}{\sigma^2} G(\mathbf{r}, \sigma^2\mathbf{I}) \quad (3.27)$$

and

$$dG(\mathbf{r}_1, \mathbf{A}\mathbf{A}^T\sigma^2)/d\mathbf{r}_1 = -(\mathbf{A}\mathbf{A}^T\sigma^2)^{-1}\mathbf{r}_1 G(\mathbf{r}_1, \mathbf{A}\mathbf{A}^T\sigma^2) \quad (3.28)$$

Equation (3.26 looks different from the first moment equation (3.21) because the Gaussian has been normalized before differentiation. Convolving with second derivatives of a Gaussian gives

$$F_1(\mathbf{r}) * d^2G(\mathbf{r}, \sigma^2\mathbf{I})/d(\mathbf{r}\mathbf{r}^T) = \mathbf{A}^T F_2(\mathbf{r}_1) * d^2G(\mathbf{r}_1, \mathbf{A}\mathbf{A}^T\sigma^2)/d(\mathbf{r}_1{\mathbf{r}_1}^T)\mathbf{A} \tag{3.29}$$

where

$$d^2G(\mathbf{r}, \sigma^2\mathbf{I})/d(\mathbf{r}\mathbf{r}^T) = \frac{(\mathbf{r}\mathbf{r}^T)/\sigma^2 - \mathbf{I}}{\sigma^2} G(\mathbf{r}, \sigma^2\mathbf{I}) \tag{3.30}$$

and

$$d^2G(\mathbf{r}_1, \mathbf{A}\mathbf{A}^T\sigma^2)/d(\mathbf{r}_1{\mathbf{r}_1}^T) = [(\mathbf{A}^T\mathbf{A}\sigma^2)^{-1}\mathbf{r}_1\mathbf{r}_1^T(\mathbf{A}^T\mathbf{A}\sigma^2)^{-1} - (\mathbf{A}\mathbf{A}^T\sigma^2)^{-1}]G(\mathbf{r}_1, \mathbf{A}\mathbf{A}^T\sigma^2) \tag{3.31}$$

The form of the moment equations is due to the linearity of the affine transform. For example the fact that there is an $\mathbf{A}^{-1}$ multiplying the first moment on the right hand side in 3.21 is directly due to the linearity of the affine transform. Gaussian derivatives are products of polynomial functions with Gaussians. Thus filtering with Gaussian derivatives is equivalent to computing a linear combination of Gaussian weighted moments. It is no surprise, therefore, that the first derivative of the Gaussian equation (equation 3.26) is similar in form to the first moment equation. The first derivative on the right side of 3.26 is pre–multiplied by $\mathbf{A}^T$. Similarly, equations (3.29) and (3.23) are seen to be closely related; the difference is the additional term due to the Gaussian weighting in equation (3.29) and due to normalization.

## 3.4 Incorporating Translation

Translation can be incorporated in the above equations in a straightforward way. When translation is included the brightnesses are related as follows:

$$F_1(\mathbf{r}) = F_2(\mathbf{A}\mathbf{r} + \mathbf{t}) \tag{3.32}$$

This implies that the Gaussian equations when translation is included are given by

$$\begin{aligned} \int F_1(\mathbf{r})G(\mathbf{r},\sigma^2\mathbf{I})d\mathbf{r} &= \int F_2(\mathbf{Ar}+\mathbf{t})G(\mathbf{Ar},\mathbf{R}\Sigma\mathbf{R}^T)d(\mathbf{Ar}) \quad (3.33) \\ &= \int F_2(\mathbf{Ar})G(\mathbf{Ar}-\mathbf{t},\mathbf{R}\Sigma\mathbf{R}^T)d(\mathbf{Ar}) \quad (3.34) \end{aligned}$$

This may be rewritten in the notation previously used as

$$F_1 * G(\mathbf{r},\sigma^2\mathbf{I}) = F_2 * G(\mathbf{Ar}-\mathbf{t},\mathbf{R\Sigma R}^T) \quad (3.35)$$

Similarly the first derivative equation is given by

$$F_1 * G'(\mathbf{r},\sigma^2\mathbf{I}) = \mathbf{A}^T F_2 * G'(\mathbf{Ar}-\mathbf{t},\mathbf{R\Sigma R}^T) \quad (3.36)$$

and the second derivative equation by

$$F_1 * G''(\mathbf{r},\sigma^2\mathbf{I}) = \mathbf{A}^T F_2 * G''(\mathbf{Ar}-\mathbf{t},\mathbf{R\Sigma R}^T)\mathbf{A} \quad (3.37)$$

The translation tends to interact with the deformation terms as will be seen later (Chapter 5).

### 3.4.1 Moments in Other Domains

Similar moment equations can be written in the frequency domain. In the frequency domain, translation is encoded by the phase. By taking the moments of the magnitude of the frequency response, the moments can be made insensitive to small variations in translation (gross translation still needs to be recovered). The disadvantage of frequency based methods lies in the need for large windows to obtain good frequency resolution.

Taking the Fourier transform of equation 3.1 results in:

$$S_1(\mathbf{f}) = S_2(\mathbf{A^{-1}f})/\det(A) \tag{3.38}$$

where $S_1(f)$ and $S_2(f)$ are the Fourier transforms of $F_1$ and $F_2$ respectively. Thus the zeroth moment of the magnitude of the frequency response (which is identical to the square root of the power spectral density) is given by

$$\int_{\infty}^{\infty} |S_1(\mathbf{f})| d\mathbf{f} = \int_{\infty}^{\infty} |S_2(\mathbf{A}^{-1}\mathbf{f})| d(\mathbf{A}^{-1}\mathbf{f}) \tag{3.39}$$

Using more compact notation, this may be written as

$$\mu_{01} = \mu_{02} \tag{3.40}$$

where $\mu$ is the moment of the square root of the power spectral density, and the first subscript denotes the order of the moment. It is straightforward to show that the first moment equation is given by

$$\mu_{11} = \mathbf{A}^{-1}\mu_{12} \tag{3.41}$$

and the second moment equation by

$$\mu_{21} = \mathbf{A}^{-1}\mu_{22}(\mathbf{A}^{-1})^T \tag{3.42}$$

Unlike the moment equations in the spatial domain, the determinant terms are no longer present.

Using ordinary moments (as opposed to Gaussian–weighted moments) implies that all the frequencies are equally weighted for the zeroth moment, while the higher frequencies are much more heavily weighted for the first and second moments. This is not desirable since noise often affects the high frequency response much more than the low frequency

response. It is, therefore, advantageous to weight the low frequencies more than the high frequencies. The desired weighting may be achieved by using Gaussian–weighted moments in the frequency domain. The procedure for deriving Gaussian–weighted moments in the frequency domain is similar to the procedure used for deriving Gaussian–weighted moments in the spatial domain. For example the Gaussian–weighted zeroth moment in the frequency domain is given by

$$\int |S_1(\mathbf{f})| G(\mathbf{f}, \sigma^2\mathbf{I}) d\mathbf{f} = \int |S_2(\mathbf{A}^{-1}\mathbf{f})| G(\mathbf{A}^{-1}\mathbf{f}, \mathbf{R}\mathbf{\Sigma}\mathbf{R}^T) d(\mathbf{A}^{-1}\mathbf{f}) \det(\mathbf{A}^{-1}) \tag{3.43}$$

Compared with the Gaussian equations for brightness there is an additional determinant term which complicates the computation of the affine transform.

Identical moment equations can be written using the auto-correlation of the functions $F_1$ and $F_2$ instead of the functions themselves; the auto-correlation of a function F is defined as

$$L(\mathbf{r}) = \int_{-\infty}^{\infty} F(\mathbf{r}_1) F(\mathbf{r} + \mathbf{r}_1) d\mathbf{r}_1 \tag{3.44}$$

Frequency domain methods have for the most part been confined to the shape from texture literature and have not been much used in motion work.

## 3.5 Spatial Constraints

The discussion so far has centered on filtering with the Gaussian and Gaussian derivative equations at a single point. Just using the filter outputs from a single point is not a robust strategy to recover the general affine transform. For example, Werkhoven and Koenderink [75] use the Gaussian and the Gaussian derivatives up to the second order to produce six equations for six unknowns (**A**and **t**) (see Chapter 5). However, as their experiments [76] and our experiments (see Chapter 5) show, this system of equations is not robust - essentially because all measurements are made around a single point (that is, all the derivatives are estimated at the same point). To provide robustness, the spatial constraint that the

deformation is approximately unchanging over some neighborhood can be imposed. Note that by making this region large the affine transform can be better approximated. However, if the region is too large it may not be possible to model it using a single affine transform, either because the underlying surface is non-planar or because over a large enough region the effects of perspective projection must be modelled.

An affine transform models the deformation over a region rather than at a single point. Robust recovery of the affine transform requires that measurements be made over the entire region rather than at a single point. Measurements made over a small area around the center of the region may provide an approximation to the affine transform (which may be poor). This occurs because discretization limits the precision with which measurements can be made. The following 1-D example demonstrates that using measurements over a small area near the center can produce poor results.

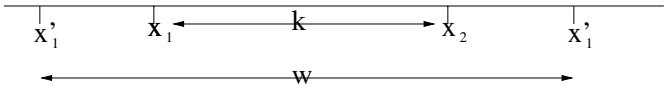



**Figure 3.3.** Precision of measurements.

Consider two points $x_1$ and $x_2$ separated by a distance of k pixels (see Figure 3.3. Assume that discretization limits the accuracy with which the distance can be measured to m (say $m \leq 1$) pixels. Now let there be a scale change s so that the points $x_1'$ and $x_2'$ are now separated by w = ks pixels. The scale change will, therefore be given by w/k. However, the distances k and w can only be measured with finite precision. Thus the measured scale change is given by round(w/m)/round(k/m) where round(x) returns the value of x rounded to the nearest integer. Some numbers will help to clarify this result. Let m = 0.5, k = 2, w = 2.3. Then the actual scale = w/k = 2.3/2 = 1.15. The measured scale is, however, 1.25 which is far from 1.15. Assume that m stays the same but k = 6 and w = 6.9. In the second case, the measured scale = 1.17 which is much closer to the correct value. The example demonstrates that large errors can occur if the spacing between the points where

the measurements are made is small. Although the example was demonstrated using points, a similar problem is encountered when using filter based methods.

On the other hand, if the region size is too large, it may no longer be possible to model the deformations using an affine transform. Thus there is a limit to how large such regions can be made.

The spatial constraints can be enforced by

1. applying one or more of the constraint equations at more than one point in a neighborhood or

2. by using filters at multiple scales and orientations.

It is also possible to combine them and use multiple scales and apply the constraint equations at each point. However, this does not seem to have any specific advantage since the scale information is already present.

The second approach takes advantage of the fact that image information is present at multiple scales. When the scale of a filter is increased, the filter provides information from many more pixels. A filter used at a single scale averages the information over the pixels. However, it is the spatial variation of the signal which helps determine the affine transform. By using multiple scales, averages over different sized windows are used providing much more information about the spatial variation of the signal. It is straightforward to modify the equations to use multiple scales. The Gaussian and Gaussian derivative equations above are valid for any arbitrary choice of $\sigma$. Thus by selecting different values of $\sigma$, filters at different scales are obtained.

The other approach is to use the filter at many points in a region. Filtering at a point $l_i$ modifies the generalized Gaussian equation to give

$$\int F_1(\mathbf{r})G(\mathbf{r}-\mathbf{l}_i,\sigma^2\mathbf{I})d\mathbf{r} = \int F_2(\mathbf{Ar})G(\mathbf{A}(\mathbf{r}-\mathbf{l}_i),\mathbf{R\Sigma R}^T)d(\mathbf{Ar}) \tag{3.45}$$

Thus is the image is filtered at point $\mathbf{A}\mathbf{l}_i$ in the second image patch. This is equivalent to introducing a translation equal to $\mathbf{l}_i$. The first derivative equation is given by

$$\int F_1(\mathbf{r})G'(\mathbf{r}-\mathbf{l}_i,\sigma^2\mathbf{I})d\mathbf{r} = \int A^T F_2(\mathbf{A}\mathbf{r})G'(\mathbf{A}(\mathbf{r}-\mathbf{l}_i),\mathbf{R}\mathbf{\Sigma}\mathbf{R}^T)d(\mathbf{A}\mathbf{r}) \tag{3.46}$$

while the second derivative equation is

$$\int F_1(\mathbf{r})G''(\mathbf{r}-\mathbf{l}_i,\sigma^2\mathbf{I})d\mathbf{r} = \int \mathbf{A}^T F_2(\mathbf{A}\mathbf{r})G''(\mathbf{A}(\mathbf{r}-\mathbf{l}_i),\mathbf{R}\mathbf{\Sigma}\mathbf{R}^T)d(\mathbf{A}\mathbf{r})\mathbf{A} \tag{3.47}$$

When a system of equations is obtained by filtering with circular Gaussians over many scales but at a single point, the rotational invariance of the system is preserved. The main disadvantage of filtering at many points in a region is that the system of equations is no longer rotationally invariant.

# CHAPTER 4

# SOLUTION FOR THE SIMILARITY CASE

## 4.1 Introduction

In the previous chapter, the problem of matching two image patches under an affine deformation was formulated. It was shown there that if Gaussian filters were used, then the outputs were equal provided the Gaussian was affine transformed in the same manner as the image. Thus, the problem of finding the affine transform was, therefore, reduced to the problem of finding the deformation of a known function, the Gaussian.

In this chapter, the Gaussian equation is solved for the specific case when the affine transform is expressed as a similarity transform, i.e. a scale change $s$, a rotation $\mathbf{R}$ and a translation ($\mathbf{A} = s\mathbf{R} + \mathbf{t}$). When the affine transform can be expressed as a similarity transform, the deformed Gaussian has a standard deviation $s * \sigma$ (see Chapter 3) where $s$ is the scale of the similarity transform. Since the affine transform is unknown, the correct deformed Gaussian may be be recovered by searching the space of all deformed Gaussians. This search is expensive, so an alternative is proposed here. The alternative involves sampling the space of deformed Gaussians and linearly interpolating between them.

As discussed in the previous chapter, an overconstrained system of equations can be produced either by varying the scale of the Gaussian equations or by formulating the equation at more than one point in a region (or by a combination of both). The first approach is used in this chapter to solve for similarity transforms. The next chapter shows how the second approach may be used to recover general affine transforms.

Similarity transforms arise in a number of practical situations. For example, when a robot's motion is mostly translational with small rotational components and the structures

in the image are shallow (i.e. have small extent in depth compared to their average depth), the deformation can be adequately explained by a scale change proportional to the depth of the corresponding structure and a rotation [57]. Matching under similarity transforms is also useful for image retrieval. Consider, for example, the problem of searching for logos or trademarks in a database. The logos or trademarks are obtained from scanned paper documents and therefore their orientation and aspect ratio are likely to be unchanged. The only unknowns are the relative scaling of the logos or trademarks and their position on the document. This can be modelled by a similarity transform.

### 4.1.1 Solving the Zeroth Moment Equation

Equation 3.11 will be initially solved assuming that the translation is zero and then modified to deal with translation. When the affine transform is described by A = s **R**, equation (3.11) can be written as

$$F_1(\mathbf{r}) * G(\mathbf{r}, \sigma^2) = F_2(\mathbf{r}_1) * G(\mathbf{r}_1, (\sigma^*)^2) \tag{4.1}$$

where $\sigma^* = s\sigma$. The important point here is that the rotation matrix does not figure in $\sigma^*$. The problem of finding the scale parameter has therefore been converted into the problem of finding the value of $\sigma^*$. The present formulation reduces the problem of correpondence to finding the appropriate affine parameters of a known function - the Gaussian. Older methods [6, 12, 28] have instead concentrated on the much more difficult problem of trying to find the correspondences between the functions $F_1$ and $F_2$. Additional simplicity will be obtained from the fact that the affine transform can be analytically interpolated.

The equation can be solved by sampling the space of possible values of $\sigma^*$, filtering for each sampled value and declaring the solution to be that value of $\sigma^*$ for which the above equation is satisfied. Such high resolution sampling is expensive. A more elegant approach uses the fact that the affine transform can be analytically interpolated. The idea is to sample

coarsely over a small set of $\sigma^*$ and then interpolate using a Taylor series approximation. Consider first a given $\sigma^*$. The Taylor series approximation to first order gives

$$\begin{aligned} G(\mathbf{r}_1, (s\sigma)^2) &\approx G(\mathbf{r}_1, \sigma^2) + \alpha\sigma \frac{\partial G(\mathbf{r}_1, \sigma^2)}{\partial \sigma} && (4.2) \\ &= G(\mathbf{r}_1, \sigma^2) + \alpha\sigma^2 \nabla^2 G(\mathbf{r}_1, \sigma^2) && (4.3) \end{aligned}$$

where $s = 1 + \alpha$. The last equality follows from the diffusion equation $\frac{\partial G}{\partial \sigma} = \sigma \nabla^2 G$. This allows the convolution (4.1) to be written as

$$F_1(\mathbf{r}) * G(\mathbf{r}, \sigma^2) \approx F_2(\mathbf{r}_1) * G(\mathbf{r}_1, \sigma^2) + \alpha\sigma^2 F_2(\mathbf{r}_1) * \nabla^2 G(\mathbf{r}_1, \sigma^2) \qquad (4.4)$$

Equation 4.4 is linear in $s$ or $\alpha$. To find s, three filtering operations need to be performed: two Gaussian filtering operations and one Laplacian operation. Note that the above equation expresses the well-known result that a Laplacian can be approximated by a difference of Gaussians.

#### 4.1.1.1 Issues of Scale

Information in an image is scale dependent. There may be information present at several different scales or at only one of them. A method which does not take issues of scale into account is not likely to be robust. Thus it is desirable to solve the above equation at several different scales ($\sigma$'s). Let a set of $\sigma_i$'s be chosen (see below on the choice of $\sigma_1$'s). For each such $\sigma_i$ an equation of the form (4.4) may be written giving the following system of equations

$$\begin{aligned} F_1 * G(\mathbf{r}, \sigma_0^2) &\approx F_2 * G(\mathbf{r}_1, \sigma_0^2) + \alpha\sigma_0^2 F_2 * \nabla^2 G(\mathbf{r}_1, \sigma_0^2) \\ F_1 * G(\mathbf{r}, \sigma_1^2) &\approx F_2 * G(\mathbf{r}_1, \sigma_1^2) + \alpha\sigma_1^2 F_2 * \nabla^2 G(\mathbf{r}_1, \sigma_1^2) \\ \ldots\ldots &\approx \ldots\ldots \\ F_1 * G(\mathbf{r}, \sigma_i^2) &\approx F_2 * G(\mathbf{r}_1, \sigma_i^2) + \alpha\sigma_i^2 F_2 * \nabla^2 G(\mathbf{r}_1, \sigma_i^2) \end{aligned}$$

$$\begin{aligned} ..... &\approx ..... \\ F_1 * G(\mathbf{r}, \sigma_l^2) &\approx F_2 * G(\mathbf{r}_1, \sigma_l^2) + \alpha\sigma_l^2 F_2 * \nabla^2 G(\mathbf{r}_1, \sigma_l^2) \end{aligned} \tag{4.5}$$

This is an over-constrained set of equations in the unknown $\alpha$. The redundancy offered by the over-constrained problem makes it more robust with respect to noise.

The choice of $\sigma_i's$ is mostly arbitrary although some general criteria may be specified. Gaussian filtering may be viewed as a local average computed using Gaussian weighting. A value of $\sigma_i$ that is too small will, therefore, make the system sensitive to noise. On the other hand, good localization requires that $\sigma_i$ not be too large. The actual values are not crucial. In practice, a set of eight different $\sigma's$ were chosen. They were all equally spaced apart by half an octave (a factor of 1.4). Since the filters need to range from $-4\sigma$ to $4\sigma$ (see Chapter 6 for a discussion) the filter width = 8 $\sigma$. The widths actually chosen were (3,5,7,10,14,20,28,40) (see also Jones and Malik [27]).

The above system of equations was cast into the following linear least squares function

$$\mathbf{\Sigma}_i || F_1 * G(\mathbf{r}, \sigma_i^2) - F_2 * G(\mathbf{r}_1, \sigma_i^2) + \alpha\sigma_i^2 F_2 * \nabla^2 G(\mathbf{r}_1, \sigma_i^2) ||^2 \tag{4.6}$$

and was solved using Singular Value Decomposition (SVD). It was found that the lowest filters (widths = 3,5,7) were noisy and hence they were disregarded (one reason may be that the Laplacian is noisy when the filter size is small). The scale was recovered fairly accurately using the other widths (see Section 4.1.2 for details). This set of filter widths worked better than another one where 8 filters were used with their $\sigma's$ spaced apart by a factor of 1.2. Gaussian and Gaussian derivative filters which are closely related in scale are much more correlated than those which are further apart in scale. Thus if the number of filters is the same, the filters spaced apart by a factor of 1.2 provide less information than those spaced 1.4 apart.

#### 4.1.1.2 Choosing a Different Operating Point

For large scale changes (say scale change $\geq 1.2$) the recovered scale tends to be poor. This is because the Taylor series approximation is good only for small values of $\alpha$. The advantage of the current approach is that the linearization point can be shifted i.e. the right-hand side of (3.11) can be linearized with respect to a $\sigma$ different from the one on the left-hand side. As we have pointed out earlier, other methods linearize the function F or the Gaussian with respect to $\mathbf{r}$ and are therefore constrained to measuring small affine transforms. Let the right-hand side of (4.4) be linearized around $\sigma_j$ to give the following equation

$$F_1 * G(., \sigma_i^2) \approx F_2 * G(., \sigma_j^2) + \alpha' \sigma_j^2 F_2 * \nabla^2 G(., \sigma_j^2) \tag{4.7}$$

$\alpha'$ is solved for using the above equation. The scale change $s$ is then computed from $\alpha'$ using the expression $(s = \sigma_j / \sigma_i (1 + \alpha'))$. The strategy therefore is to pick different values of $\sigma_j$ and solve (4.7) ( or actually an overconstrained version of it). Each of these $\sigma_j$ will result in a value of $\alpha'$. The correct value of $\alpha'$ is that which is most consistent with the equations. For example, let the correct scale be 1.3, and assume that the $\sigma_j$ values chosen are $(\sigma, 1.4\sigma, 2.0\sigma)$. Let the values of $s$ recovered using these $\sigma_j$ be (1.25,1.32,1.5) respectively. Then 1.32 is consistent with expanding around $1.4\sigma$ in the sense of being closest to its expansion point. By choosing the $\sigma_j$ appropriately, it can be ensured that no new convolutions are required.

In principle, arbitrary scale changes can be recovered using this technique. The range of scale changes required depends on the application. For example, when a robot is moving down a hallway, most of the scale changes are less than or equal to 2.4. Since each operating point can cover up to $(1.2 * \sigma_i)$, three operating points $(\sigma, 1.4\sigma, 2.0\sigma)$ should suffice. In other applications like aerial imagery or the detection of logos, the scale changes will be much larger and more operating points may be needed.

#### 4.1.1.3 Finding Image Translation

Image translation (i.e. optic flow) can be recovered in the following manner. Let $F_1$ and $F_2$ be similarity transformed versions of each other (i.e. they differ by a scale change, a rotation and a translation). Assume that an estimate of the translation $\mathbf{t}_0$ is available. Linearizing with respect to $\mathbf{r}$ and $\sigma$ gives

$$F_1(\mathbf{r}+\mathbf{t}_0)*G(\mathbf{r},\sigma^2)-\delta\mathbf{t}^T F_1(\mathbf{r}+\mathbf{t}_0)*G(\mathbf{r},\sigma^2) \approx F_2*G(.,\sigma^2)+\alpha\sigma^2 F_2*\nabla^2 G(.,\sigma^2) \quad (4.8)$$

which is again linear in both the scale and the residual translation $\delta\mathbf{t}$. As before an over-constrained version of this equation using multiple scales is obtained and solved for the unknown parameters. Large scales are handled as before.

The value of $\mathbf{t}_0$ can be obtained either by a local search or from a coarser level in a multi–resolution pyramid scheme, while $\delta\mathbf{t}$ is estimated from the equation.

Note that since the Gaussians are rotation invariant, the translation can be recovered for arbitrary rotations about an axis perpendicular to the image. Standard schemes which are derived from correlation are not able to do this (see the next section for an example).

### 4.1.2 Experimental Results

A number of experiments were carried out using the algorithm. The first experiment involved creating a synthetic image. This synthetic image was a cosine wave generated by the equation $F_1(x,y) = 127cos(\omega_y y)$ with $\omega_y = 0.2$. An interesting feature of this image is that there is no information along the y direction (the so-called aperture problem). In spite of that the scale can be recovered. A second cosine function was generated using the following function $F_2(x,y) = 127cos(s\omega_y x + \phi_y)$. $F_2$ is rotated 90 deg. with respect to $F_1$ and also scaled by the factor $s$. For various values of $s$, the scale was recovered using the zeroth moment. The results are tabulated in Table 4.1. The experiment was repeated with noise added. First, uniform noise ranging from -10 to 10 was added to $F_2$. Second, Gaussian noise with a standard deviation of 10 was added to $F_2$. These results are also

**Table 4.1.** Recovered scales for cosine images.

| Actual | No Noise | | | Gaussian Noise $\sigma$ = 10 | | | Uniform Noise (-10,10) | | |
|---|---|---|---|---|---|---|---|---|---|
| | scale s | * 100 | | scale s | * 100 | | scale s | * 100 | |
| | | $\frac{\delta s}{s}$ | $\frac{\delta s}{s-1}$ | | $\frac{\delta s}{s}$ | $\frac{\delta s}{s-1}$ | | $\frac{\delta s}{s}$ | $\frac{\delta s}{s-1}$ |
| 1.05 | 1.050 | 0 | 0 | 1.040 | 1.0 | 20 | 1.040 | 1.0 | 20 |
| 1.10 | 1.101 | 0.09 | 0.01 | 1.082 | 1.6 | 18 | 1.098 | 0.2 | 2.0 |
| 1.15 | 1.158 | 0.7 | 5.3 | 1.152 | 0.1 | 1.3 | 1.148 | 0.1 | 1.3 |
| 1.20 | 1.213 | 1.1 | 6.5 | 1.198 | 0.2 | 1.0 | 1.192 | 0.7 | 4.0 |
| 1.40 | 1.408 | 0.6 | 2.0 | 1.415 | 1.1 | 3.8 | 1.427 | 1.9 | 6.8 |
| 1.60 | 1.639 | 2.4 | 6.5 | 1.630 | 1.9 | 5.0 | 1.591 | 0.6 | 1.5 |
| 1.80 | 1.855 | 3.1 | 6.9 | 1.838 | 2.1 | 4.8 | 1.816 | 0.9 | 2.0 |

tabulated in Table 4.1. Two operating points were used: $\sigma$ and $1.4\sigma$. The appropriate operating point was picked as discussed earlier in the text.

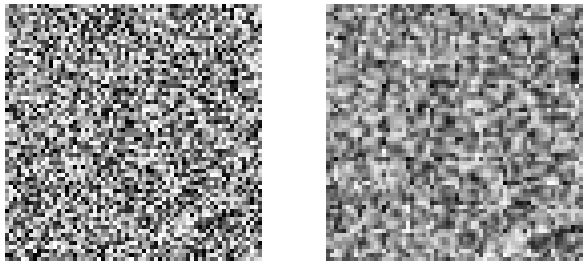
Table 4.1 is to be read as follows. The first column in Table 4.1 is the actual scale while column 2 shows the recovered scale in the noise-free case. Two different percentage errors are tabulated and they arise from the following considerations. Assume that an object is at a depth of $z_0$ and after a translation $T_z$ in the z direction, its new depth is $z_1 = z_0 + T_z$. Then the percentage error in finding the quantity $z_1/z_0$ is given by $\frac{\delta s}{s} * 100$ and this is tabulated in column 3 for the noise-free case. On the other hand, the percentage error in finding the quantity $T_z/z_0$ is given by $\frac{\delta s}{s-1} * 100$) and this is tabulated for the noise-free case in column 4. Which of these quantities is more important? Since the depth $z_0$ is apriori unknown, the quantity of relevance at least in the motion case is $T_z/z_0$ and the corresponding percentage error is more significant. Similar values are tabulated when Gaussian noise (columns 5,6 and 7) and uniform noise (columns 8,9, and 10) are added .

The results are excellent in the noise-free case. The percentage errors in column 3 are all less than about 3% while even in column 4 the percentage errors do not exceed 7% (the average errors are of course much lower). Note that the method recovers scale accurately in spite of the large rotation.

**Table 4.2.** Recovered scales for random dot images.

| Actual | Recovered Scale s | $\frac{\delta s}{s} * 100$ | $\frac{\delta s}{s-1} * 100$ |
|---|---|---|---|
| 1.05 | 1.059 | 0.9 | 18 |
| 1.10 | 1.104 | 0.4 | 4.0 |
| 1.15 | 1.138 | 0.8 | 8.0 |
| 1.20 | 1.180 | 1.7 | 10 |
| 1.40 | 1.398 | 0.1 | 0.5 |
| 1.60 | 1.569 | 1.99 | 5.2 |
| 1.80 | 1.722 | 4.3 | 9.8 |

With noise added, the results are as good except for the lowest scales (1.05 and 1.10). One possible method of improving the results for the lower scales is to use filters separated by ratios smaller than 1.4.



(a) First Image (b) Second Image

**Figure 4.1.** Random dot images. The second image is similarity transformed with respect to the first.

The experiment was repeated with random dot images. A random dot image of size 64 by 64 was generated (see Figure 4.1(a)). The image was then affine transformed and smoothed using a cubic interpolation scheme (see Figure 4.1(b)). For various values of the scale factor $s$, the scale was recovered using the zeroth moment method. The results are tabulated in Table 4.1.2. The highest error in column 4 is less than 9% if the smallest scale (1.05) is ignored. Again the average error is much lower.

Figure (4.2) illustrates the usefulness of this algorithm. A random dot image is scaled by a factor of 1.1 and rotated around an axis perpendicular to the image by 30 deg. For comparison, the optic flow on the left was produced by using Anandan's SSD based pyramid

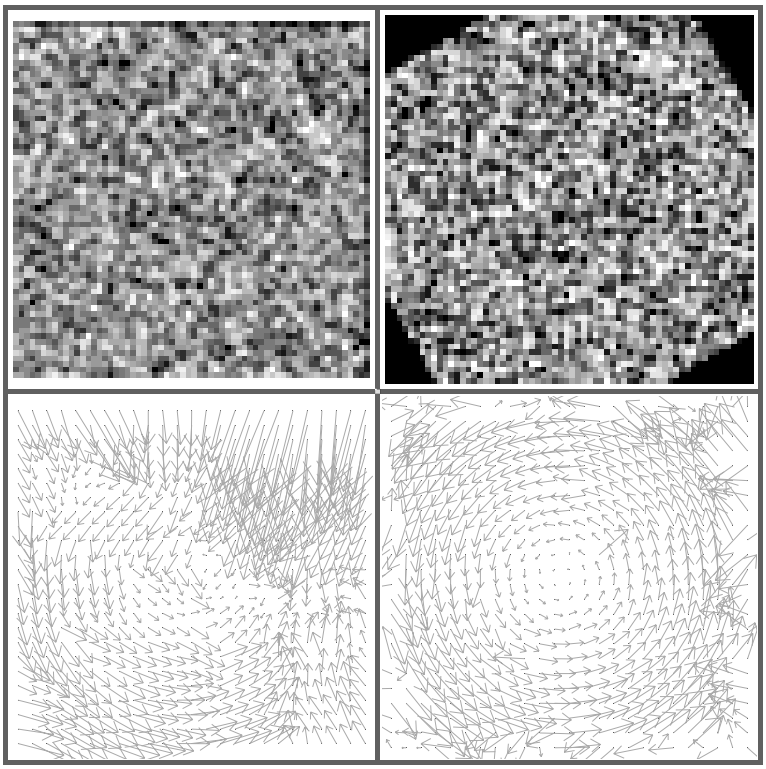


**Figure 4.2.** Random dot sequence

scheme [3]. Note that the algorithm fails quite dramatically because of the large rotation. This occurs because for correct matching the template also needs to be rotated by the same angle. The SSD algorithm is created on the assumption that the deformation is small and that the corresponding points in both images have not moved a lot. For small angles, the template rotation can be ignored, but this cannot be done for large rotations. On the other hand the results of running the algorithm described here are shown on the right-hand side. The flow shown is clearly rotational. *Note that the flow has been computed at every point without fitting a global model*. To the best of our knowledge no other existing algorithm can compute the flow correctly in this situation. A histogram of the recovered scale values peaks at 1.1 which is the correct value.

**Figure 4.3.** Dollar bill

Figure (4.3) shows a dollar bill scaled by 1.4. The algorithm correctly recovers the scale as 1.41.

# CHAPTER 5

# SOLVING FOR THE GENERAL AFFINE TRANSFORM

The general affine transform between two image patches is specified by six parameters: two translation and four deformation parameters. The increase in the number of parameters, compared to the similarity case, has the following consequence: the technique used for solving for similarity transforms does not work any more. That is, just using the Gaussian equations at a single point over many scales does not suffice.

The affine transform may, however, be recovered by using additional constraint equations. This may be done in a number of ways, either by using the derivative equations, or by using additional equations at additional points or by a combination of both. In the following sections, all these approaches will be developed.

First, it is shown that the Gaussian equation can be linearized with respect to the affine parameters. A similar approach can be used to linearize the derivatives of the Gaussian. By filtering with the Gaussian and the first two derivatives of the Gaussian and then linearizing the resulting equations with respect to the affine parameters, a linear system of equations is obtained. It is shown, experimentally, that this linear system, first solved by Werkhoven and Koenderink [75, 76], produces poor results (see Section 5.2.1). The poor performance of the Werkhoven and Koenderink algorithm is due to a number of reasons. The two main reasons are:

1. They attempt to solve a non-linear equation by approximating it with a linear equation and then solving for the resulting equations.

2. A minimal number of equations (six) are used. In practice, in the presence of noise, this number of equations does not sufficiently constrain the equations.

A number of procedures may be used to solve these problems and produce better algorithms. Better linear approximations may be obtained by using coarse sampling and linearizing the parameters with respect to the coarse samples. A solution which is closer to the original non-linear equation may be obtained by iteratively refining the linear solution. More equations may be obtained by using multiple scales, and solving the Gaussian and Gaussian derivative equations at many points simultaneously. The use of these techniques leads to two specific algorithms called GauArea and DGauArea. Experiments conducted on these algorithms (see Chapter 6) show that they give good results.

## 5.1 Linearizing The Gaussian

The simplest approach is to linearize the Gaussian and its derivatives with respect to the affine parameters as follows.

The Gaussian equation 3.17 is given by:

$$F_1(\mathbf{r}) * G(\mathbf{r}, \sigma) = F_2(\mathbf{Ar}) * G(., \sigma^2 \mathbf{A}\mathbf{A}^T) \tag{5.1}$$

With translation included, this becomes

$$F_1(\mathbf{r}) * G(\mathbf{r}, \sigma) = F_2(\mathbf{Ar} + \mathbf{t}) * G(\mathbf{Ar}, \sigma^2 \mathbf{A}\mathbf{A}^T) \tag{5.2}$$

which may be rewritten as:

$$F_1(\mathbf{r}) * G(\mathbf{r}, \sigma) = F_2(\mathbf{Ar}) * G(\mathbf{Ar} - \mathbf{t}, \sigma^2 \mathbf{A}\mathbf{A}^T). \tag{5.3}$$

The Gaussian on the right hand side is given by:

$$G(\mathbf{r}_1, \sigma^2 \mathbf{A}\mathbf{A}^T) = \frac{1}{(2\pi) det(\mathbf{A}) \sigma^2} exp(-\frac{\mathbf{r}_1^T (\mathbf{A}\mathbf{A}^T)^{-1} \mathbf{r}_1}{2\sigma^2}) \tag{5.4}$$

It is straightforward to show using the above equation that

$$det(\mathbf{A}^{-1})G(\mathbf{A}^{-1}\mathbf{r}_1, \sigma) = G(\mathbf{r}_1, \sigma^2\mathbf{A}\mathbf{A}^T) \tag{5.5}$$

Thus equation 5.3 can, therefore, be rewritten as:

$$F_1(\mathbf{r}) * G(\mathbf{r}, \sigma) = F_2(\mathbf{r}_1) * G(\mathbf{A}^{-1}(\mathbf{r}_1 - \mathbf{t}), \sigma)det(\mathbf{A}^{-1}). \tag{5.6}$$

If $\mathbf{B} = \mathbf{A} - \mathbf{I}$ is small, the $\mathbf{A}$ inverse inside the Gaussian may be linearly approximated to give

$$\begin{aligned} G((\mathbf{I}+\mathbf{B})^{-1}(\mathbf{r}_1 - \mathbf{t}), \sigma) &\approx G((\mathbf{I}-\mathbf{B})(\mathbf{r}_1 - \mathbf{t}), \sigma) \\ &\approx G((\mathbf{r}_1 - \mathbf{B}\mathbf{r}_1 - \mathbf{t}), \sigma) \\ &\approx G(\mathbf{r}_1, \sigma) - (\mathbf{B}\mathbf{r}_1 + \mathbf{t})^T G'(\mathbf{r}_1, \sigma) \end{aligned} \tag{5.7}$$

i.e.

$$\begin{aligned} G(\mathbf{A}^{-1}(\mathbf{r}_1 - \mathbf{t}), \sigma) &\approx G(\mathbf{r}_1, \sigma) - [(b_{11}x_1 + b_{12}y_1 + t_x)G_{x_1}(\mathbf{r}_1, \sigma) \\ &+ (b_{21}x_1 + b_{22}y_1 + t_y)G_{y_1}(\mathbf{r}_1, \sigma)] \end{aligned} \tag{5.8}$$

where the $b_{ij}$ are elements of $\mathbf{B}$, $\mathbf{t} = (t_x, t_y)$ and $\mathbf{r}_1 = (x_1, y_1)$. Using the following identities

$$-xG_x = \sigma^2 G_{xx} + G \tag{5.9}$$

$$-yG_y = \sigma^2 G_{yy} + G \tag{5.10}$$

$$-yG_x = -xG_y = \sigma^2 G_{xy} \tag{5.11}$$

equation 5.8 may be rewritten as

$$G(\mathbf{A}^{-1}(\mathbf{r}_1 - \mathbf{t}), \sigma) \approx G(\mathbf{r}_1, \sigma) + \sigma^2[(b_{11}G_{x_1x_1}(\mathbf{r}_1, \sigma) + b_{12}G_{x_1y_1}(\mathbf{r}_1, \sigma)$$

$$
\begin{aligned}
&+ \quad (b_{21} G_{x_1 y_1}(\mathbf{r}_1, \sigma) + b_{22} G_{y_1 y_1}(\mathbf{r}_1, \sigma)] \\
&- \quad t_x G_{x_1}(\mathbf{r}_1, \sigma) - t_y G_{y_1}(\mathbf{r}_1, \sigma) \\
&+ \quad (b_{11} + b_{22}) G(\mathbf{r}_1, \sigma). \qquad (5.12)
\end{aligned}
$$

Substituting the above expression in equation 5.6 and simplifying gives:

$$
\begin{aligned}
F_1(\mathbf{r}) * G(\mathbf{r}, \sigma) - F_2(\mathbf{r}_1) * G(\mathbf{r}_1, \sigma) \quad &\approx \quad \sigma^2 [(b_{11} F_2 * G_{x_1 x_1}(\mathbf{r}_1, \sigma) + b_{12} F_2 * G_{x_1 y_1}(\mathbf{r}_1, \sigma) \\
&+ \quad (b_{21} F_2 * G_{x_1 y_1}(\mathbf{r}_1, \sigma) + b_{22} F_2 * G_{y_1 y_1}(\mathbf{r}_1, \sigma)] \\
&- \quad t_x F_2 * G_{x_1}(\mathbf{r}_1, \sigma) - t_y F_2 * G_{y_1}(\mathbf{r}_1, \sigma). \qquad (5.13)
\end{aligned}
$$

Note: the approximation $det(\mathbf{A}^{-1}) \approx 1 - b_{11} - b_{22}$ has been used in the above equation.

Thus, when an image patch and the affine transformed version of the image patch are filtered with Gaussians of the same standard deviation, the difference between the two filter outputs may be approximated by the expression given on the RHS. The expression on the RHS is a sum of terms where each term is the product of an an affine parameter with the second image filtered by a Gaussian derivative. The right hand side is linear in the affine parameters and contains derivatives of Gaussians up to order 2.

## 5.2 Linearizing Gaussian Derivatives

Gaussian derivatives may similarly be linearized. This is done here by adapting a method used by Werkhoven and Koenderink [75].

Let

$$
\phi_{n-m,m}(\mathbf{r}, \sigma) \equiv \sigma^n \frac{d^n G(\mathbf{r}, \sigma)}{dx^{n-m} dy^m} \qquad (5.14)
$$

Note that the Gaussian derivatives have been normalized so that they all have the same energy. Energy normalization can be done for circular Gaussian derivatives by multiplying

each derivative by $\sigma^n$ where n is the order of the derivative. Now differentiating 5.6 with respect to x (n-m) times and with respect to y m times gives:

$$\int F_1(\mathbf{r})\phi_{n-m,m}(\mathbf{r},\sigma)d\mathbf{r} = \int F_2(\mathbf{r}_1)\phi_{n-m,m}(A^{-1}(\mathbf{r}_1-\mathbf{t}),\sigma)d\mathbf{r}_1 det(\mathbf{A}^{-1}). \quad (5.15)$$

As before if $\mathbf{B} = \mathbf{A} - \mathbf{I}$ is small, then

$$\begin{aligned}
\phi_{n-m,m}(A^{-1}(\mathbf{r}_1-\mathbf{t}),\sigma) &\approx \phi_{n-m,m}((\mathbf{I}+\mathbf{B})^{-1}(\mathbf{r}_1-\mathbf{t}),\sigma) \\
&\approx \phi_{n-m,m}((\mathbf{I}-\mathbf{B})(\mathbf{r}_1-\mathbf{t}),\sigma) \\
&\approx \phi_{n-m,m}((\mathbf{r}_1-\mathbf{B}\mathbf{r}_1-\mathbf{t}),\sigma) \\
&\approx \phi_{n-m,m}(\mathbf{r}_1,\sigma) - [\phi_{n-m+1,m}(\mathbf{r}_1,\sigma)[b_{11}x+b_{12}y+t_x] \\
&+ \phi_{n-m,m+1}(\mathbf{r}_1,\sigma)[b_{21}x+b_{22}y+t_y]]/\sigma \quad (5.16)
\end{aligned}$$

The above equation may be rewritten using the following recurrence equations A.10,A.11 (see Appendix A for a derivation).

$$\begin{aligned}
x\phi_{n-m,m}(\mathbf{r}_1,\sigma) &= -\sigma(n-m)\phi_{n-m-1,m}(\mathbf{r}_1,\sigma) - \sigma\phi_{n-m+1,m}(\mathbf{r}_1,\sigma) \quad (5.17)\\
y\phi_{n-m,m}(\mathbf{r}_1,\sigma) &= -\sigma m\phi_{n-m,m-1}(\mathbf{r}_1,\sigma) - \sigma\phi_{n-m,m+1}(\mathbf{r}_1,\sigma) \quad (5.18)
\end{aligned}$$

Applying the above recurrence relations to equation 5.16 gives

$$\begin{aligned}
\phi_{n-m,m}(A^{-1}(\mathbf{r}_1-\mathbf{t}),\sigma) &\approx \phi_{n-m,m}(\mathbf{r}_1,\sigma) \\
&+ b_{11}[(n-m+1)\phi_{n-m,m}(\mathbf{r}_1,\sigma) + \phi_{n-m+2,m}(\mathbf{r}_1,\sigma)] \\
&+ b_{12}[m\phi_{n-m+1,m-1}(\mathbf{r}_1,\sigma) + \phi_{n-m+1,m+1}(\mathbf{r}_1,\sigma)] \\
&+ b_{21}[(n-m)\phi_{n-m-1,m+1}(\mathbf{r}_1,\sigma) + \phi_{n-m+1,m+1}(\mathbf{r}_1,\sigma)] \\
&+ b_{22}[(m+1)\phi_{n-m,m}(\mathbf{r}_1,\sigma) + \phi_{n-m,m+2}(\mathbf{r}_1,\sigma)]
\end{aligned}$$

$$+ \quad (t_x \phi_{n-m+1,m}(\mathbf{r}_1, \sigma) + t_y \phi_{n-m,m+1}(\mathbf{r}_1, \sigma))/\sigma \tag{5.19}$$

Using $det(\mathbf{A}^{-1}) \approx 1 - b_{11} - b_{22}$ and substituting the above expression in equation 5.15 gives

$$\begin{aligned}
&\int F_1(\mathbf{r})\phi_{n-m,m}(\mathbf{r}, \sigma)d\mathbf{r} - \int F_2(\mathbf{r}_1)\phi_{n-m,m}(\mathbf{A}^{-1}(\mathbf{r}_1 - \mathbf{t}), \sigma)d\mathbf{r}_1 \\
&= \quad b_{11}[(n-m)q_{n-m,m} + q_{n-m+2,m}] \\
&+ \quad b_{12}[mq_{n-m+1,m-1} + q_{n-m+1,m+1}] \\
&+ \quad b_{21}[(n-m)q_{n-m-1,m+1} + q_{n-m+1,m+1}] \\
&+ \quad b_{22}[mq_{n-m,m} + q_{n-m,m+2}] \\
&+ \quad (t_x q_{n-m+1,m} + t_y q_{n-m,m+1})/\sigma
\end{aligned} \tag{5.20}$$

where $q_{n-m,m} = \int F_2(\mathbf{r}_1)\phi_{n-m,m}(\mathbf{r}_1, \sigma)d\mathbf{r}_1$

The left hand side is again the difference between $F_1$ and $F_2$ after they have been filtered with a Gaussian derivative. The right hand side is again the sum of terms where each term is the product of a Gaussian derivative and an affine parameter. Note that the order of the Gaussian derivative on the right hand side is at most 2 more than that on the left hand side.

Werkhoven and Koenderink [75] used equation 5.20 to linearize the Gaussian and the first and second derivatives at a single scale to obtain the following set of six equations in six unknowns.

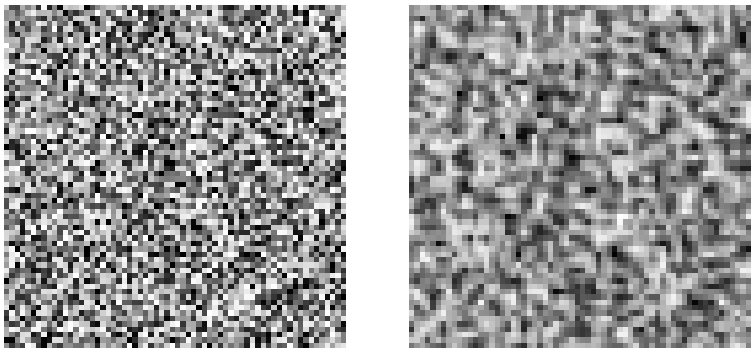
$$\begin{bmatrix} l_{00}-q_{00} \\ l_{10}-q_{10} \\ l_{01}-q_{01} \\ l_{20}-q_{20} \\ l_{11}-q_{11} \\ l_{02}-q_{02} \end{bmatrix} \approx \begin{bmatrix} -q_{10} & q_{20} & q_{11} & -q_{01} & q_{11} & q_{02} \\ -q_{20} & q_{10}+q_{30} & q_{21} & -q_{11} & q_{01}+q_{21} & q_{12} \\ -q_{11} & q_{21} & q_{10}+q_{12} & -q_{02} & q_{12} & q_{01}+q_{03} \\ -q_{30} & 2q_{20}+q_{40} & q_{31} & -q_{21} & 2q_{11}+q_{31} & q_{22} \\ -q_{21} & q_{11}+q_{31} & q_{20}+q_{22} & -q_{12} & q_{02}+q_{22} & q_{11}+q_{13} \\ -q_{12} & q_{22} & 2q_{11}+q_{13} & -q_{03} & q_{13} & 2q_{02}+q_{04} \end{bmatrix} \begin{bmatrix} t_x/\sigma \\ b_{11} \\ b_{12} \\ t_y/\sigma \\ b_{21} \\ b_{22} \end{bmatrix} \tag{5.21}$$

where $l_{ij} = \int F_1 \phi_{i,j} d\mathbf{r}$. The above equation is linear in the unknowns $t_i, b_{jk}$.

Werkhoven and Koenderink experimented with the set of equations above and their conclusion [76] noted that the errors were low if the affine transformation consisted of a single deformation e.g. it consisted only of one of the following transformations - a scale change, a shear, a rotation or a translation. If, however, the affine transformation was composed of more than one of the above transformations, they found that for small relative deformations (of the order of 1%), the errors were about 3-6 % in the presence of the other transformations. On the other hand, they found that for a 1% relative translation (relative to the scale of the Gaussian) the errors in the recovered translation were as high as 50 % in the presence of other deformations. In the presence of noise, the performance was worse.

### 5.2.1 Experiments with the Werkhoven and Koenderink Algorithm

A 1% relative deformation is quite small. For many purposes, larger deformations need to be recovered. In practical situations, the Werkhoven and Koenderink algorithm performs quite poorly for large deformations and even for moderate translations. We have carried out experiments using the Werkhoven and Koenderink algorithm. Figures 5.2(a)-5.2(f) show the performance of the Werkhoven-Koenderink algorithm as the deformation is increased. These experiments were carried out by scaling one image with respect to the other in the presence of other transformations (the other transformations were kept small). The performance for other deformations is not plotted here, but shows the same general trend.



(a) First Image (b) Second Image

**Figure 5.1.** Test images for Werkhoven-Koenderink algorithm. The second image is an affine transformed version of the first. The affine transformation lies in the middle of the deformation range in the graphs. The second image has been scaled with respect to the first in the presence of other transformations; see text.

The figures show the results of an experiment conducted with two random dot images (for examples see Figures 5.1(a) and 5.1(b)). Both images were of size 64 by 64 and the intensity values varied between 0 and 255. The first image was created by choosing the intensities uniformly between 0 and 255. The second image was obtained by affine warping the first image about its center and then adding independent identically distributed (IID) Gaussian noise. Two of the affine parameters $b_{12}$ and $b_{21}$ were kept fixed at 0.1. Since performance under scaling is desired, the other two affine parameters $b_{11}$ and $b_{22}$ were set equal to each other and varied from 0.0 to 1.1 in steps of 0.1 (i.e. $a_{11}$ and $a_{22}$ were varied from 1.0 to 2.1).The variance of the noise for the results displayed in the figures was 30 (similar results are obtained for less noise). The translation was kept fixed at (0.5,0.5) pixels. Figure 5.1(a) and Figure 5.1(b) show an example pair of images. In this particular case, the affine deformation was set to $a_{11} = a_{22} = 1.5$

Figures 5.2(a)-5.2(f) display the RMS error in the recovered value of the affine parameters $b_{11}, b_{12}, b_{21}, b_{22}, t_x, t_y$ as the affine parameters $(b_{11}, b_{22})$ were varied (a single affine transform was computed for each pair of images). The RMS error was computed by conducting each experiment 100 times and adding random noise with a different seed each

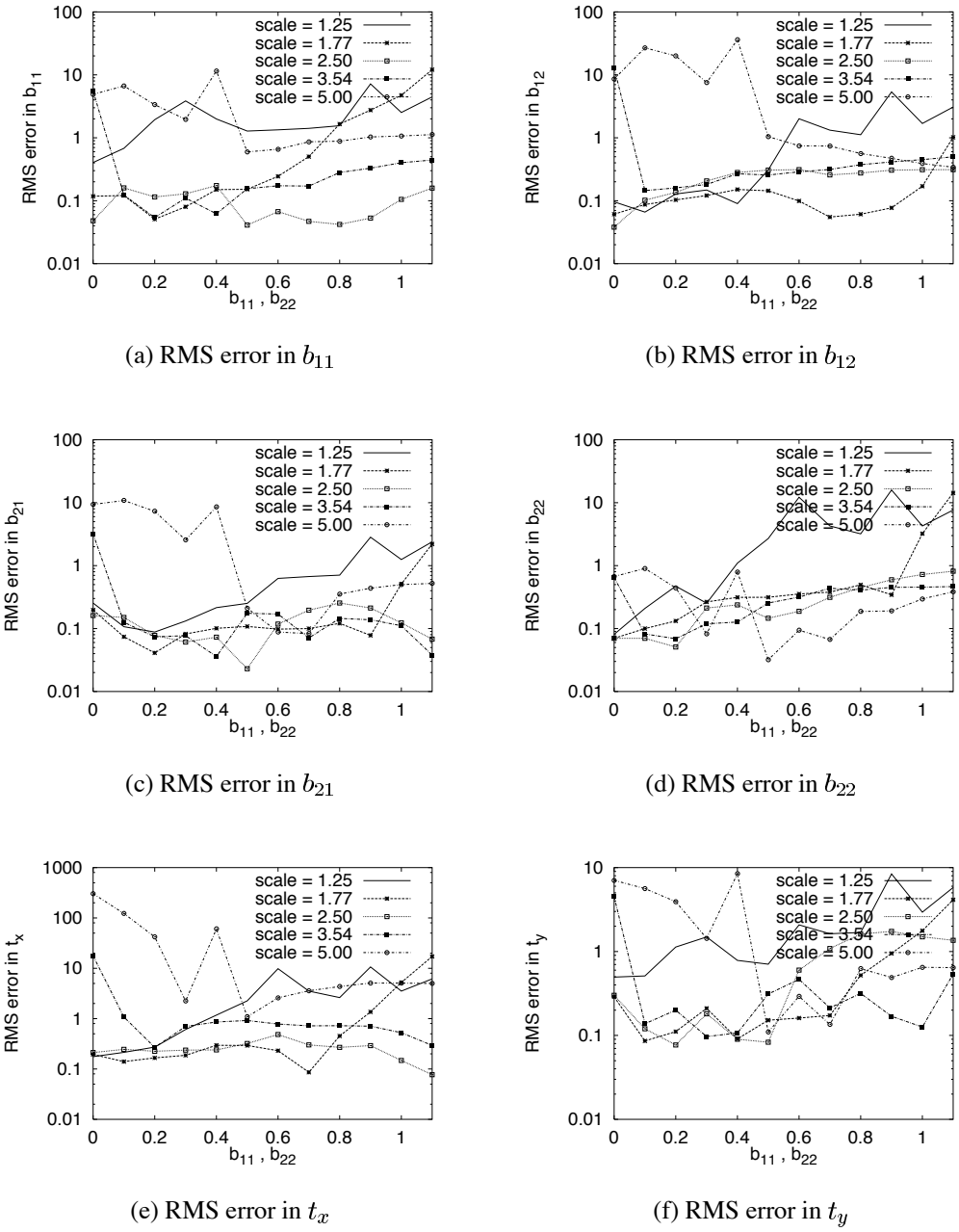
time. The RMS error is plotted using a log scale. The experiment was conducted using five different scales for the Gaussian; adjacent scale steps differed by a factor of $\sqrt{2}$.

At the smallest scale used, 1.25, the performance is really poor. As the scale of the Gaussian is increased, the performance improves. This improvement is due to the increase in the size of the filter support. However, at large scales, performance again goes down. This may be in part due to poorer localization in the translation parameters. The poorer performance is more prominent for small values of $b_{11}, b_{22}$ (this is visible in the plots for scale 5.0). A second effect is due to the fact that Gaussian filtering blurs the image. If the scale of the Gaussian is large, the blurring may cause the image to appear uniform. Random dot images, are in particular, susceptible to such blurring because most of their energies reside at high frequencies.

As the scale of the Gaussian is increased from 1.25, performance improves. For scale 1.77, the performance is good for low values of $(b_{11}, b_{22})$ but rapidly deteriorates for larger values of $(b_{11}, b_{22})$ for most of the parameters. This is to be expected since the linearized derivatives are a poor approximation for large affine deformations. For scale (2.5) the performance is also good for larger values of $b_{11}, b_{22}$. For scale (3.54), the performance deteriorates at low values of $b_{11}, b_{22}$ and this is again seen at scale (5.0). To some extent, the scale dependence of the results is to be expected since information in the image is scale dependent.

Figure 5.3 replots the same RMS errors in terms of percentages. At low values of $(b_{11}, b_{22})$ the % error is large because percentages are computed by dividing by $b_{11}$. However, even for larger values of $(b_{11}, b_{22})$ the **best** results show an RMS error of roughly 10 % . This is a large error.

The number of equations equals the number of unknowns in the Werkhoven-Koenderink algorithm. Thus, an equation count seems to imply that the linear system of equations solved by Werkhoven-Koenderink is sufficiently constrained. However, as the experiments above show, the algorithm produces large errors in the recovered affine parameters. This




(a) RMS error in $b_{11}$

(b) RMS error in $b_{12}$

(c) RMS error in $b_{21}$

(d) RMS error in $b_{22}$

(e) RMS error in $t_x$

(f) RMS error in $t_y$

**Figure 5.2.** Performance of the Werkhoven-Koenderink algorithm for large deformations as a function of $b_{11}$

implies that, in practice, the equations do not constrain the problem sufficiently and using more equations might help. We will use more equations to constrain the problem and improve results.

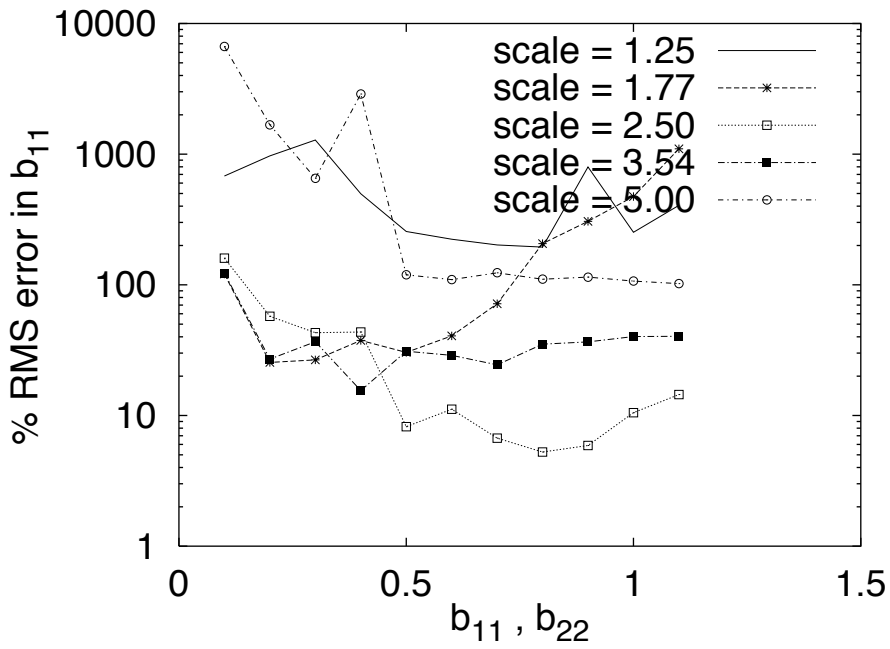



**Figure 5.3.** RMS percentage error in $b_{11}$ as a function of scale change for the Werkhoven-Koenderink algorithm

## 5.3 Strategies for Improving the Recovery of the Affine Parameters

The poor performance of the Werkhoven and Koenderink algorithm is due to a number of reasons. The two main reasons are:

1. They attempt to solve a non-linear equation by approximating it with a linear equation and then solving for the resulting equations.

2. A minimal number of equations (six) are used. In practice in the presence of noise, this number of equations does not sufficiently constrain the equations.

Algorithms much more robust than Werkhoven and Koenderink's can be created by using several different strategies. These include

1. The use of multiple scales to provide more equations.

2. Iterative refinement of the solution. The linear solution is iteratively refined to produce a solution closer to the solution of the original non-linear equation.

3. The use of coarse sampling to handle larger affine transforms.

4. The linearization of higher order derivatives to provide additional equations.

5. Linearization at additional points in the image which again produces more equations.

These strategies will now be discussed. Werkhoven and Koenderink [76] did not consider any of these options.

### 5.3.1 Use of Multiple Scales

Werkhoven and Koenderink used a single scale to compute the affine transform. However, information in an image is spread over a large number of scales. These scales provide additional information to constrain the solution. Thus, by using multiple scales, the results can be improved. Essentially the same set of equations derived earlier is obtained at each scale and a least mean squares formulation is set up. This is then solved to obtain the affine transform.

### 5.3.2 Iterative Refinement

The equations are linearized about the origin (i.e. the origin of the space of affine transforms) with the implicit assumption that the affine transforms between the image are small. Larger affine transforms may be solved for by linearizing the equations about a point close to the correct (i.e. actual) affine transform. This can be done by either iteratively refining the solution or by coarsely sampling the space of affine transforms. Iterative refinement can be done in at least two diffferent ways:

1. The estimate of the affine transform obtained at each stage may be used to warp the first image. The residual affine transform is then computed between the second image and the warped version of the first image. This process is refined (repeated) for a certain number of iterations.

2. Equivalently, the first image is refilterd with an elliptical Gaussian whose parameters are determined by the current estimate of the affine transform. The residual affine transform is then computed between the second image and the refiltered version of the first image.

Elliptical Gaussians are not separable except when they are aligned with the x and y axes. Filtering with them is, therefore, an expensive operation. Although warping is also an expensive operation, it is cheaper than filtering with elliptical Gaussians. Iterative refinement using warping is, therefore, the strategy adopted here.

### 5.3.3 Coarse Sampling

The results may also be improved by coarsely sampling the space of affine transforms and then linearizing the equations.

The expression in 5.20 is valid if $\mathbf{B}$ is small. It is straightforward to show that this linearization can also be done when the $\sigma$ of the Gaussian derivative, about which the right hand side of equation 5.15 is linearized, is different from that on the left hand side. i.e.

$$\begin{aligned}
&\int F_1(\mathbf{r})\phi_{n-m,m}(\mathbf{r},\sigma)d\mathbf{r} - \int F_2(\mathbf{r}_1)\phi_{n-m,m}(\mathbf{r}_1,\sigma_2)d\mathbf{r}_1 \\
&= [(1+b_{11})\sigma_2/\sigma - 1][(n-m)q_{n-m,m} + q_{n-m+2,m}] \\
&+ b_{12}(\sigma_2/\sigma)[mq_{n-m+1,m-1} + q_{n-m+1,m+1}] \\
&+ b_{21}(\sigma_2/\sigma)[(n-m)q_{n-m-1,m+1} + q_{n-m+1,m+1}] \\
&+ [(1+b_{22})\sigma_2/\sigma - 1][mq_{n-m,m} + q_{n-m,m+2}] \\
&+ (t_x q_{n-m+1,m} + t_y q_{n-m,m+1})/\sigma_2 \qquad (5.22)
\end{aligned}$$

where $q_{n-m,m} = \int F_2(\mathbf{r})\phi_{n-m,m}(\mathbf{r},\sigma_2)d\mathbf{r}$ and $\sigma_2 \neq \sigma$. One advantage of this formulation is that when the affine transform has large scale changes (but not large rotations or shears), this expression may still be used to make $\mathbf{B}/\sigma$ small even if $\mathbf{B}$ is not small.

Thus a portion of the affine space maybe sampled by using different values of $\sigma_2$ for the right hand side. Note the resulting linearized Gaussian derivatives are still circular. It is not clear whether the expressions can be linearized in terms of elliptical Gaussian derivatives (essentially because it is not clear whether the recurrence relations still hold).

However, it is also possible to sample the space of affine transforms by warping the first image using a set of predetermined affine parameters. For most practical purposes this set can be small. For example, to allow for large rotations, one can warp the first image in steps of 22.5 degrees. When the affine transform is solved for, the correct sample is the one for which the lowest residual error is obtained. This procedure has been implemented and used to solve for large rotations.

### 5.3.4 Linearization of Higher Order Derivatives

Additional equations may also be obtained by using higher order derivatives and then linearizing them. They can be obtained from equation 5.20 by substituting for the appropriate order of the derivatives.

Higher order derivatives do provide additional equations although there are some disadvantages to using them. They tend to be more sensitive to noise. Although (in 1D) the first and second derivative equations are uncorrelated with each other, derivatives of higher orders are correlated with them and, therefore, do not provide much additional information. Thus, higher order derivatives will not be considered further here.

### 5.3.5 Linearization at Additional Points

In the derivations above it is assumed that the Gaussian filtering is done at the origin of the coordinate system (wherever it is chosen to be). Filtering at a point $\mathbf{l}_i$ modifies the generalized gaussian equation 3.17 as follows:

$$\int F_1(\mathbf{r})G(\mathbf{r}-\mathbf{l_i},\sigma^2\mathbf{I})d\mathbf{r} = \int F_2(\mathbf{Ar})G(\mathbf{A}(\mathbf{r}-\mathbf{l_i})),\sigma^2\mathbf{A}\mathbf{A}^T)d(\mathbf{Ar}) \tag{5.23}$$

Thus if the image is filtered at point $\mathbf{l}_i$ in the first image patch, it must be filtered at point $\mathbf{A}\mathbf{l}_i$ in the second image patch. Note that this is similar to moving the translation point.

An overconstrained set of equations can be obtained from equation 5.23 by linearizing it and choosing a number of different points $\mathbf{l}_i$. The approach used in Section 5.1 may be used to linearize equation 5.23. Here, a somewhat different approach will be used to linearize this equation. The right hand side of equation 5.23 is first linearized about the point $\mathbf{l}_i$ to give:

$$\begin{aligned}\int F_1(\mathbf{r})G(\mathbf{r}-\mathbf{l_i},\sigma^2\mathbf{I})d\mathbf{r} &\approx \int F_2(\mathbf{A}\mathbf{r})G(\mathbf{A}\mathbf{r}-\mathbf{l_i}),\sigma^2\mathbf{A}\mathbf{A}^T)d(\mathbf{A}\mathbf{r}) \\ &- [(\mathbf{A}-\mathbf{I})\mathbf{l}_i]^T\int F_2(\mathbf{A}\mathbf{r})G'(\mathbf{A}\mathbf{r}-\mathbf{l_i}),\sigma^2\mathbf{A}\mathbf{A}^T)d(\mathbf{A}\mathbf{r})\end{aligned} \tag{5.24}$$

where G' is the derivative of G with respect to the image coordinates. The next step is to approximate the deformed gaussian. Now

$$G(.,\sigma^2\mathbf{A}\mathbf{A}^T) = \frac{1}{(2\pi)^{n/2}det(\mathbf{A})\sigma}exp(-\frac{\mathbf{r}_1^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{r}_1}{2\sigma^2}) \tag{5.25}$$

If $\mathbf{B}=\mathbf{A}-\mathbf{I}$ is small, the $\mathbf{A}$ inverses inside the exponential may be linearly approximated to give

$$G(.,\sigma^2\mathbf{A}\mathbf{A}^T) \approx \frac{1}{(2\pi)^{n/2}det(\mathbf{A})\sigma}exp(-\frac{\mathbf{r}_1^T(\mathbf{B}-\mathbf{I})(\mathbf{B}-\mathbf{I})^T\mathbf{r}_1}{2\sigma^2}). \tag{5.26}$$

This may now be linearized with respect to the elements of $\mathbf{B}$ to give

$$G(.,\sigma^2\mathbf{A}\mathbf{A}^T) \approx G(.,\sigma)+\sigma^2[b_{11}G_{xx}(.,\sigma)+b_{12}G_{xy}(.,\sigma)+b_{21}G_{xy}(.,\sigma)+b_{22}G_{yy}(.,\sigma)] \tag{5.27}$$

where the $b_{ij}$ are elements of $\mathbf{B}$. Considering only the linear terms in $\mathbf{B}$, equation (5.24) can therefore be written as as:

$$\begin{aligned} F_1 * G(\mathbf{r}-\mathbf{l}_i,\sigma) &\approx F_2 * G(\mathbf{r}_1-\mathbf{l}_i,\sigma) - (\mathbf{B}\mathbf{l}_i)^T F_2 * G'(\mathbf{r}_1-\mathbf{l}_i,\sigma) \\ &+ \sigma^2[b_{11}F_2 * G_{xx}(\mathbf{r}_1-\mathbf{l}_i,\sigma) + b_{22}F_2 * G_{yy}(\mathbf{r}_1-\mathbf{l}_i,\sigma) \\ &+ (b_{12}+b_{21})F_2 * G_{xy}(\mathbf{r}_1-\mathbf{l}_i,\sigma)]. \end{aligned} \tag{5.28}$$

The effect of the $\mathbf{l}_i$ is similar to that of a translation. Thus, with translation included, the above equation may be written as:

$$\begin{aligned} F_1 * G(\mathbf{r}-\mathbf{l}_i,\sigma) &\approx F_2 * G(\mathbf{r}_1-\mathbf{l}_i,\sigma) - (\mathbf{B}\mathbf{l}_i)^T F_2 * G'(\mathbf{r}_1-\mathbf{l}_i,\sigma) \\ &+ \sigma^2[b_{11}F_2 * G_{xx}(\mathbf{r}_1-\mathbf{l}_i,\sigma) + b_{22}F_2 * G_{yy}(\mathbf{r}_1-\mathbf{l}_i,\sigma) \\ &+ (b_{12}+b_{21})F_2 * G_{xy}(\mathbf{r}_1-\mathbf{l}_i,\sigma)] \\ &- \mathbf{t}^T F_2 * G'(\mathbf{r}_1-\mathbf{l}_i,\sigma). \end{aligned} \tag{5.29}$$

For notational purposes the above equation may be rewritten as:

$$\begin{aligned} h_{i00} - g_{i00} &= b_{11}(g_{i20} - x_i g_{i10}/\sigma) + b_{12}(g_{i11} - y_i g_{i10}/\sigma) + b_{21}(g_{i11} - x_i g_{i01}/\sigma) \\ &+ b_{22}(g_{i02} - y_i g_{i01}/\sigma) - t_x g_{i10} - t_y g_{i01} \end{aligned} \tag{5.30}$$

where $g_{imn} = \int \sigma^{m+n} F_2 G_{x^m y^n}(\mathbf{r}_1 - \mathbf{l}_i, \sigma), h_{imn} = \int \sigma^{m+n} F_1 G_{x^m y^n}(\mathbf{r}_1 - \mathbf{l}_i, \sigma), (x_i, y_i) = \mathbf{l}_i$, and $G_{x^m y^n}$ denotes that G is differentiated m times with respect to x and n times with respect to y.

The above equation is linear in the affine parameters $b_{ij}, \mathbf{t}$. A number of methods incorporate the idea of filtering at many points [6, 12, 64]. However, none of these compensate for the deformation terms. In essence the difference between the traditional linearization methods and the technique presented here are the additional second derivative terms.

The above equation may also be compared to the first row of equation (5.21) with $g_{ijk}$ and $h_{ijk}$ corresponding to $q_{jk}$ and $l_{jk}$ respectively. In equation (5.21) filtering is done only at the origin. Filtering at points other than the origin introduces the extra terms $(\mathbf{Bl}_i)^T F_2 * G'(\mathbf{r}_1 - \mathbf{l}_i, \sigma)$ i.e. it has the same effect as multiplying $(\mathbf{Bl}_i)^T$ by the coefficient of the translation term.

#### 5.3.5.1 First derivatives of a Gaussian

A similar derivation may be used to generate a linearized constraint equation for the derivatives of a Gaussian filtered at a point $\mathbf{l}_i$. Instead of using a formal derivation we note that the the effect of filtering at other points $\mathbf{l}_i$ is equivalent to a translation $\mathbf{Bl}_i$. Thus a linearized equation for filtering with any Gaussian derivative at a point $\mathbf{l}_i$ is obtained by adding a term which is equal to the dot product of $\mathbf{Bl}_i$ with the coefficient of the translation term to equation (5.20). Thus filtering the image with $G_x$ at a point $\mathbf{l}_i$ gives:

$$\begin{aligned} hi10 - gi10 &= b_{11}(g_{i10} + g_{i30} - g_{i20}x_i/\sigma) + b_{12}(g_{i21} - g_{i20}y_i/\sigma) \\ &+ b_{21}(g_{i01} + g_{i21} - g_{i11}x_i/\sigma) + b_{22}(g_{i12} - g_{i11}y_i/\sigma) \\ &- t_x g_{120} - t_y g_{i11} \end{aligned} \tag{5.31}$$

Filtering the image with the y derivative $G_y$ at a point $\mathbf{l}_i$ gives:

$$\begin{aligned} hi01 - gi01 &= b_{11}(g_{i21} - g_{i11}x_i/\sigma) + b_{12}(g_{i10} + g_{i12} - g_{i11}y_i/\sigma) + b_{21}(g_{i12} - g_{i02}x_i/\sigma) \\ &+ b_{22}(g_{i01} + g_{i03} - g_{i02}y_i/\sigma) - t_x g_{i11} - t_y g_{i02} \end{aligned} \tag{5.32}$$

## 5.4 Algorithms

The strategies discussed in the previous sections can be used to formulate practical algorithms to recover affine transforms. Two algorithms are developed in this section. GauArea is based on using the Gaussian equation at many points and DGauArea is based

on using the first derivatives of the Gaussian at many points. More algorithms can be formulated using higher derivatives of Gaussians or combinations of different orders of derivatives of Gaussians, but these will not be discussed here.

### 5.4.1 GauArea

An algorithm may be devised by using the linearized version of the Gaussian equation 5.30 at many points. Using several points $\mathbf{l}_i$, a least means square problem may be formulated as follows:

$$\begin{aligned} E^2 = & \sum_i [h_{i00} - g_{i00} - b_{11}(g_{i20} - x_i g_{i10}/\sigma) - b_{12}(g_{i11} - y_i g_{i10}/\sigma) \\ - & \; b_{21}(g_{i11} - x_i g_{i01}/\sigma) - b_{22}(g_{i02} - y_i g_{i01}/\sigma) + t_x g_{i10} + t_y g_{i01}]^2. \end{aligned} \quad (5.33)$$

The solution which minimizes the error $E^2$ with respect to the affine parameters is given by the solution of the following overconstrained matrix equation:

$$\mathbf{z} = \mathbf{M}\mathbf{b_t} \quad (5.34)$$

where the vector $\mathbf{z}$ is given by:

$$z = \begin{bmatrix} h_{100} - g_{100} \\ h_{200} - g_{200} \\ \cdots \\ h_{i00} - g_{i00} \end{bmatrix} \quad (5.35)$$

and the matrix $\mathbf{M}$ by:

$$\mathbf{M} = \begin{bmatrix} (g_{120} - x_1 g_{110}/\sigma) & (g_{111} - y_1 g_{110}/\sigma) & (g_{111} - x_1 g_{101}/\sigma) & (g_{102} - y_1 g_{101}/\sigma) & -g_{110} & -g_{101} \\ (g_{220} - x_2 g_{210}/\sigma) & (g_{211} - y_2 g_{210}/\sigma) & (g_{211} - x_2 g_{201}/\sigma) & (g_{202} - y_2 g_{201}/\sigma) & -g_{210} & -g_{201} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ (g_{i20} - x_i g_{i10}/\sigma) & (g_{i11} - y_i g_{i10}/\sigma) & (g_{i11} - x_i g_{i01}/\sigma) & (g_{i02} - y_i g_{i01}/\sigma) & -g_{i10} & -g_{i01} \end{bmatrix} \quad (5.36)$$

and $\mathbf{b_t}$ is a vector representing the affine parameters:

$$\mathbf{b_t} = \begin{bmatrix} b_{11} \\ b_{12} \\ b_{21} \\ b_{22} \\ t_x \\ t_y \end{bmatrix} \tag{5.37}$$

Equation (5.34) may be solved using Singular Value Decomposition (SVD). The solution requires the computation of the SVD of M. The resulting affine parameters are the solution of the equation (5.29).

#### 5.4.1.1 Iterative Refinement

The linear equation (5.29) is derived by linearizing a non-linear equation. By iteratively refining the solution, a better solution to the original non-linear equation (equation 5.23) is obtained. At each iteration, the values of the affine parameters obtained from the previous step (derived by solving equation (5.34)) are used to warp the first image. The warped image is then used to recompute the parameters $h_{i00}$ (the parameters $g_{ijk}$ do not change since they depend only on the second image). The value of **z** is then recomputed and used to calculate the residual affine motion between the warped first image and the second image.

The method of solution detailed above only requires a Gaussian convolution on the warped image and **z** to be recomputed. The SVD does not have to be recomputed and this is advantageous since the SVD is computationally expensive.

Small translations may also be computed by sampling the translation (sampling is expensive if the translation is large). For each sample, the quantity $h_{i00}$ and hence the vector z is recomputed and used to calculate both the affine motion and the residual error. The

SVD does not need to be recalculated for the sampled translations. Note that translations of a few pixels can be automatically recovered by the iterative refinement procedure without having to sample it.

#### 5.4.1.2 Multiple Scales

In the algorithm above, the scale of the Gaussians used is unspecified. Thus different scales may be used to give additional equations. For example, $h_{i00}$ and $g_{ijk}$ may be obtained by filtering with scale $\sigma$ while $h_{(i+1)00}$ and $g_{(i+1)jk}$ may be filtered with the scale $\sqrt{(2)}\sigma$. Thus, the algorithm naturally lends itself to a multi-scale approach.

#### 5.4.1.3 GauAreaN

A modified version of algorithm GauArea may be obtained by eliminating the second order derivative terms (the deformation terms). Such an algorithm is similar to that employed by [6] (with important differences). This modified algorithm will be called GauAreaN. The performance of GauArea will be compared with that of GauAreaN to show the importance of modelling deformations. GauAreaN is obtained by minimizing the following least mean squares error measure:

$$\begin{aligned} E^2 = & \sum_i [h_{i00} - g_{i00} - b_{11}(-x_i g_{i10}/\sigma) - b_{12}(-y_i g_{i10}/\sigma) - b_{21}(-x_i g_{i01}/\sigma) \\ - & \; b_{22}(-y_i g_{i01}/\sigma) + t_x g_{i10} + t_y g_{i01}]^2. \end{aligned} \tag{5.38}$$

Note that the only terms left are terms corresponding to the a translation of $\mathbf{Bl}_i$

### 5.4.2 DGauArea

Another algorithm DGauArea may be devised by using the linearized version of the first derivative equations (5.31) and (5.32) at many points. Using several points $\mathbf{l}_i$, a least means square problem may be formulated as follows:

$$
\begin{aligned}
E^2 = \quad & \sum_i [hi10 - gi10 \\
- \quad & b_{11}(g_{i10} + g_{i30} - g_{i20}x_i/\sigma) - b_{12}(g_{i21} - g_{i20}y_i/\sigma) \\
- \quad & b_{21}(g_{i01} + g_{i21} - g_{i11}x_i/\sigma) - b_{22}(g_{i12} - g_{i11}y_i/\sigma) - t_x g_{120} - t_y g_{i11}]^2 \\
+ \quad & [hi01 - gi01 \\
- \quad & b_{11}(g_{i21} - g_{i11}x_i/\sigma) - b_{12}(g_{i10} + g_{i12} - g_{i11}y_i/\sigma) \\
- \quad & b_{21}(g_{i12} - g_{i02}x_i/\sigma) - b_{22}(g_{i01} + g_{i03} - g_{i02}y_i/\sigma) - t_x g_{i11} - t_y g_{i02}]^2
\end{aligned}
\tag{5.39}
$$

DGauArea has a solution similar in form to that of GauArea i.e.

$$
\mathbf{z} = \mathbf{M}\mathbf{b_t} \tag{5.40}
$$

However, the vector **z** is given by:

$$
z = \begin{bmatrix} \cdots \\ h_{i10} - g_{i10} \\ h_{(i+1)01} - g_{(i+1)01} \\ \cdots \end{bmatrix} \tag{5.41}
$$

the $i^{th}$ row of matrix M by:

$$
\begin{aligned}
&(g_{i10} + g_{i30} - g_{i20}x_i/\sigma), (g_{i21} - g_{i20}y_i/\sigma), (g_{i01} + g_{i21} - g_{i11}x_i/\sigma), \\
&(g_{i12} - g_{i11}y_i/\sigma), -g_{120}, -g_{i11}
\end{aligned}
\tag{5.42}
$$

the $(i+1)^{st}$ row by:

$$
(g_{i21} - g_{i11}x_i/\sigma), (g_{i10} + g_{i12} - g_{i11}y_i/\sigma), (g_{i12} - g_{i02}x_i/\sigma),
$$

$$(g_{i01} + g_{i03} - g_{i02}y_i/\sigma), -g_{i11}, -g_{i02} \tag{5.43}$$

and $\mathbf{b_t}$ is again a vector representing the affine parameters.

As for GauArea, the solution for DGauArea can be iteratively refined. Multi-scale versions of DGauArea can also be used.

#### 5.4.2.1 DGauAreaN

As for GauArea, a modified version of DGauArea called DGauAreaN can be obtained by dropping the higher order terms. DGauAreaN is obtained, therefore, by minimizing:

$$\begin{aligned} E^2 = \quad & \sum_i [hi10 - gi10 \\ - \quad & b_{11}(-g_{i20}x_i/\sigma) - b_{12}(-g_{i20}y_i/\sigma) \\ - \quad & b_{21}(-g_{i11}x_i/\sigma) - b_{22}(-g_{i11}y_i/\sigma) - t_x g_{120} - t_y g_{i11}]^2 \\ + \quad & [hi01 - gi01 \\ - \quad & b_{11}(-g_{i11}x_i/\sigma) - b_{12}(-g_{i11}y_i/\sigma) \\ - \quad & b_{21}(-g_{i02}x_i/\sigma) - b_{22}(-g_{i02}y_i/\sigma) - t_x g_{i11} - t_y g_{i02}]^2 \end{aligned} \tag{5.44}$$

### 5.4.3 Comments on the Algorithms

The algorithms GauArea and DGauArea have different properties. It is to be expected that since GauArea uses the brightness information, it will be sensitive to changes in illumination. DGauArea on the other hand only uses derivative information and is, therefore, less likely to be sensitive to illumination changes.

We have also implemented an algorithm similar to DGauArea but derived by linearizing the second derivative of the Gaussian. From the limited experiments performed (not shown in this dissertation), it seems to perform worse then GauArea and DGauArea. Algorithms which use combinations of the Gaussian, first and second derivative equations also do not seem to perform better.

# CHAPTER 6

# EXPERIMENTS

This chapter details experiments conducted on both synthetic and real images using algorithms based on both the Gaussian and the Gaussian derivative equations. The experiments (in particular the ones on real images) demonstrate that algorithms which account for deformations perform better than those which do not.

Four different algorithms were tested. These were GauArea based on using the Gaussian equation at many points, DGauArea based on using the first derivative equation at many points and GauAreaN and DGauAreaN which are versions of the GauArea and DGauArea algorithms respectively that do not account for deformations. The synthetic experiments were conducted on a pair of random dot images. Clearly, the results depend on the choice of images. At one extreme, if the images are uniformly bright everywhere, there is no information present in the images and the affine transforms cannot be recovered. On the other hand, if the images have derivatives everywhere and these are large, the algorithms will perform well. Random dot images have information available everywhere and thus are a good indicator of the performance of the algorithms. Thus the synthetic experiments were done on random dot images. Experiments are also shown on some real images (i.e. both images were obtained using viewpoint changes of the camera).

The algorithms here do not use coarse sampling. The idea here is to characterize the basic algorithms. Adding coarse sampling adds to the range and in some cases accuracy of the algorithms. For example, we have verified that any rotation in the image plane can be recovered by coarsely sampling the rotation space at eight points 45 degrees apart. We leave to the future more detailed work on coarse sampling.

## 6.1 Overview of Experiments

The experiments tested not only the performance of the algorithms, but also the effects of some factors involved in practical implementations as well as the effect of choosing different parameters on the algorithms. Practical implementations need to take into account the discrete nature of digital images (the derivations in Chapter 5 assumed continuous images and continuous functions). To apply the algorithms discussed above to digital images, the Gaussian and its derivatives need to be discretized. Further, to save computational speed, finite versions of the filters need to be used i.e. the filters need to be truncated. Thus, two factors that need to be investigated are the appropriate technique for discretizing the Gaussian and its derivatives and how they can be truncated (i.e. what filter widths should be used).

Algorithms GauArea and DGauArea employ a wide variety of parameters and the effects of these parameters on the algorithms are also investigated. The parameters tested include the scale of the filters used, the number of filters of different scales and the window size of the filters. In addition the effect of noise is also tested on these algorithms.

It is impractical to test the algorithms on the entire 6 dimensional parameter space. Even if only the affine deformations are considered, a 4 dimensional space is involved. A practical choice is to test the algorithms by varying one of the affine parameters while holding the others constant. Here, the algorithms are tested for various scalings, rotations and shears while holding the other affine parameters constant (although some of the other affine parameters are non-zero). Thus, the effects of varying the algorithms' parameters are tested using a set of differently transformed images. The images are generated by synthetically transforming a random dot image over a range of scalings, rotations and shears. A comparison of the four algorithms GauArea, DGauArea, GauAreaN and DGauAreaN on the scalings, rotations and shears is also performed.

**Table 6.1.** List of experiments.

| Experiment | Images | Test of | Algorithms Tested | Subsection |
|---|---|---|---|---|
| Filter Widths | Synthetic | Filter Widths | GauArea, DGauArea | 6.4 |
| Filter Scale | Synthetic | Filter Scale | | 6.5 |
| Number of Scales Used | Synthetic | Number of Scales Used | | 6.6 |
| Window Size | Synthetic | Window Size | | 6.7 |
| Noise | Synthetic | Noise | | 6.8 |
| Comparisons | Synthetic | Scale, Change, Shear, Rotation and Large Transformations- | GauArea, DGauArea, GauAreaN, DGauAreaN | 6.9 |
| Face | Real | Approx., Small Transforms | | 6.11.1 |
| Monet | Real | Planar, Shear | | 6.11.2 |
| Blackboard | Real | Planar, Scale Change, Rotation | | 6.11.3 |
| Book | Real | Planar, Strong Perspective Effects, Rotation | | 6.11.4 |
| Pepsi | Real | Cylindrical, Scale Change, Shear and Rotation. | | 6.11.5 |

The performance of the four algorithms is also compared when more than one affine transformation is large. That is, they are also tested on images which simultaneously undergo large amounts of scaling, shear and rotation. These transformations are generated by a 3D transformation of a planar random dot image.

Finally, the algorithms are also tested on a set of real images which undergo large transformations. These images are of a variety of different kinds of scenes. Some of the tests are run on images of scenes with large planar objects, while others are run on images containing non-planar objects. The algorithms are, therefore, tested on a variety of different images.

Table 6.1 lists for convenience the experiments conducted and where they are to be found.

### 6.1.1 Summary of Experimental Results

Here, a brief summary of the experimental results is presented.

1. For the filter scales used in the experiments ($\sigma > 1.25$ pixels), the Gaussian and its derivatives may be discretized by sampling the continuous function at each pixel. Discretizing the Gaussian derivative by sampling the Gaussian and then convolving with a discrete version of the derivative leads to larger errors.

2. The filter truncation radius must be at least $\pm 4\sigma$, i.e. the filter widths must be at least $8\sigma$. Lower widths can lead to large errors in both qualitative and quantitative estimates of function derivatives.

3. For synthetic random dot images, the lower the filter scale the better the performance of GauArea and DGauArea in terms of RMS error. However, the range of DGauArea increases when large scales are used for both scalings and rotations. The range of GauArea increased with scale for rotations and decreased with scale for scalings. For real images, performance is poor with small filter scales (less then $\sigma = 2$). Performance then increases with filter scale up to a certain point (roughly $\sigma = 5$). The

difference is attributable to the fact that random dot images have texture everywhere while real images may often have large areas with little texture which have little information.

4. For synthetic images using more than one filter did not make any perceptible difference.

5. For both synthetic and real images, performance improved with increasing window size. For real images, the optimum window size varied between 30 x 30 pixels and 50 x 50 pixels. Larger windows may not correspond to planar objects in the scene and may, therefore, give rise to large errors.

6. For synthetic images, the best performance in terms of RMS error was shown by the derivative algorithms DGauArea and DGauAreaN. In terms of range (i.e. the range of affine parameters recovered), the best performance was shown by GauArea. GauAreaN performed poorly, especially when moderately large filter scales were used.

   With real images, the best performance (in terms of registration) was shown by DGauArea followed by GauArea. DGauAreaN came next and GauAreaN performed especially poorly.

## 6.2 Methodology for Experiments on Synthetic Images

The performance of the algorithms was tested using synthetic experiments where the experimental conditions could be controlled. Here, a description is presented of how the synthetic images and transformations were generated and the experiments were carried out.

The effects of varying the algorithm parameters and of varying the noise (i.e. the first five sets of experiments in Table 6.1) were tested with three kinds of deformations - scaling, rotation and shear. The scaling experiments were conducted by covarying $b_{11}$ and $b_{22}$ while holding the other affine parameters constant i.e. this is essentially a test of the algorithms

while the image is scaled. The rotation experiments were conducted by varying the rotation while the other affine parameters were held constant. The shear experiments were done by holding the shear angle constant while the shear deformation was varied. Although only one parameter was varied in the above experiments, small values of the other parameters were used.

The four algorithms were also compared when more than one kind of large deformation is present at the same time. The details of how the images were constructed, the affine parameters varied and the errors computed are described in the next few sections.

### 6.2.1 Experimental Details

In all cases both images were of size 64 by 64. For the random dot images the first image was generated using a uniform random variable and the intensity values varied between 0 and 255. The second image was obtained by affine warping the first image and then adding independent identically distributed (IID) Gaussian noise. The variance of the noise for the results displayed in the figures was 40 (similar results are obtained for less noise). The signal to noise ratio (SNR) was computed as

$$SNR = 10log_{10}(\frac{variance\ signal}{variance\ noise}) \tag{6.1}$$

For this particular image and a noise variance of 40, the SNR is 49 dB. This SNR value is typical of many CCD cameras.

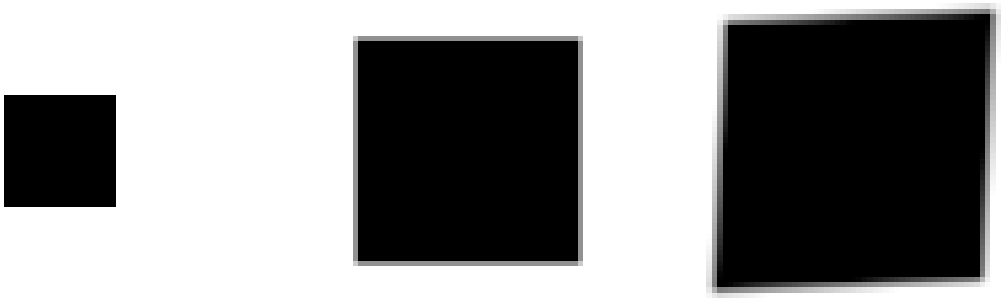
Unless otherwise specified, all synthetic experiments with the algorithms GauArea, DGauArea, GauAreaN and DGauAreaN were carried out using a window of size 13 by 13 [1] and every point within this window. Two scales were used, differing by a factor of $\sqrt{2}$ i.e. scales 1.25 and $1.25 \times \sqrt{2}$ were simultaneously used.

[1] All window size measurements and filter scales are in terms of pixel units.

The details of how the second image was affine warped for scaling, rotation and shear experiments will be listed.

#### 6.2.1.1 Scaling Experiments

For the scaling experiments, the translation was kept fixed at (0.5,0.5). Two of the affine parameters $b_{12}$ and $b_{21}$ were kept fixed at 0.1. The other two affine parameters $b_{11}$ and $b_{22}$ were set equal to each other and varied from 0.0 to 1.1 in steps of 0.1 (that is $a_{11}$ and $a_{22}$ were varied from 1.0 to 2.1). Essentially this amounts to scaling the image. Figure 6.1 shows what happens to a black square as it undergoes the transformation. Figure 6.1a shows the initial square, 6.1b shows the effects of an intermediate transformation on the square and 6.1c shows the square as it appears when the maximum scaling is applied to it.



(a) Original image (b) Mid-range scale change (c) Maximum scale change

**Figure 6.1.** Examples of scaling

#### 6.2.1.2 Rotation Experiments

For the second set of experiments, the translation was kept fixed at (0.5,0.5). The second image was scaled by a factor of 1.2 and rotated i.e. the transformation used was

$$\mathbf{A} = s\mathbf{R}(\theta) \tag{6.2}$$

(a) Original image (b) Maximum Rotation

**Figure 6.2.** Examples of rotation

where $\mathbf{R}(\theta)$ is the 2 by 2 rotation matrix given by:

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}. \tag{6.3}$$

The rotation angle $\theta$ was varied in steps of 5 deg from -45 degrees to 45 degrees. Figure 6.2 shows what happens to a black square as it undergoes the transformation. Figure 6.2a shows the initial square, 6.2b shows the effects of the maximum rotation.

#### 6.2.1.3 Shear Experiments

For the fourth set of experiments, the translation was kept fixed at (0.5,0.5). The shear applied to the second image is specified by the following affine transformation [75]:

$$\mathbf{A} = \mathbf{I} + 1/2def\mathbf{R}(\delta) \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \mathbf{R}^{-1}(\delta) \tag{6.4}$$

where $def$ is the shear deformation and $\delta$ is the shear angle. The shear angle $\delta$ was kept constant at 30 degrees. The shear deformation $def$ was varied from 0 to 0.60 in steps of 0.04. The shear may be interpreted as "a contraction in a certain direction and an expansion

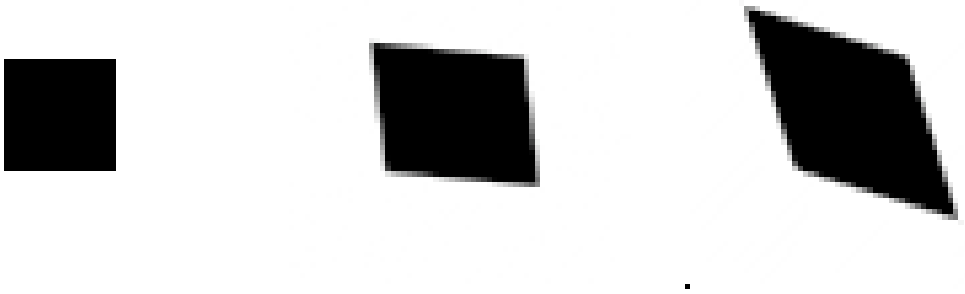
in the orthogonal direction with the axis of contraction and expansion determined by $\delta$" [75].

(a) Original image (b) Mid-range shear (c) Maximum shear

**Figure 6.3.** Examples of shear

Figure 6.3 shows what happens to a black square as it undergoes a shear transformation. Figure 6.3a shows the initial square, 6.3b shows the effects of a transformation in the middle of the range on the square and 6.3c shows the square as it appears when the maximum amount of shear is applied (a staircase edge appears on the figure because of aliasing).

#### 6.2.1.4 Large Simultaneous Transformations in More than One Parameter



(a) Original image (b) Mid-range (c) Maximum

**Figure 6.4.** Examples of images with simultaneous changes in scale, slant and tilt.

The previous experiments allowed for a large variation of one parameter and small variations of the other parameters. In this set of experiments, a number of parameters are co-varied simultaneously and by large amounts. The translation was again fixed at (0.5,0.5). Assume that the initial image is created from a fronto-parallel surface and the second image from a plane inclined to the camera at slant $\alpha$ and tilt $\beta$ and scaled s times with respect to the first plane. Then assuming weak perspective projection, there is an affine transformation between the two planes which is given by [9]:

$$\mathbf{A} = s\mathbf{R}(\beta) \begin{pmatrix} 1.0 & 0 \\ 0 & \cos(\alpha) \end{pmatrix} \mathbf{R}^{-1}(\beta) \tag{6.5}$$

The slant $\alpha$ was varied from 0 deg. to 56.6 deg ($\cos(\alpha)$ was varied from 1.0 to 0.55 in steps of 0.05). The scale s was simultaneously co-varied from 1.0 to 1.9 in steps of 0.1 and the angle $\beta$ was co-varied from 0 deg. to 45 deg. in steps of 5 deg.

Figure 6.4 shows what happens to a black square transformed in the above manner. Figure 6.4a shows the initial square, Figure 6.4b shows the effects of a transformation in the middle of the range on the square and Figure 6.4c shows the square as it appears at the end of the range.

### 6.2.2 Presentation and Interpretation of Results

The performance of the algorithms is displayed using the root mean square (RMS) error for each of the affine parameters as a function of the deformation. Percentage errors are not plotted as these are misleading. When computing percentage errors, they are large for low values of the affine parameters. This happens because percentage errors are computed by taking the ratio of the error to the value of the affine parameter. For small values of the affine parameters, the percentage errors are, therefore, large. On the other hand, the RMS error is roughly constant over a large range of affine parameters, thus establishing that the varying percentage error is an artifact of computation. The RMS error was computed by

conducting each experiment 30 times and adding Gaussianly distributed random noise with a different seed each time.

The results are evaluated in terms of both the RMS error and the range of the algorithms. The range of the algorithms will be defined here as that set of deformations for which the RMS error in all affine deformation parameters (i.e. $b_{11}, b_{12}, b_{21}, b_{22}$) is less than 0.1. For convenience only graphs for the affine parameter $b_{11}$ will be shown in the main body of the text. In most cases, RMS errors for the other affine parameters show similar trends. In situations where a different affine deformation parameter would lead to a smaller range, the graph for it is displayed in place of the graph for $b_{11}$. For completeness the rest of the graphs for the affine parameters are displayed in Appendix C.

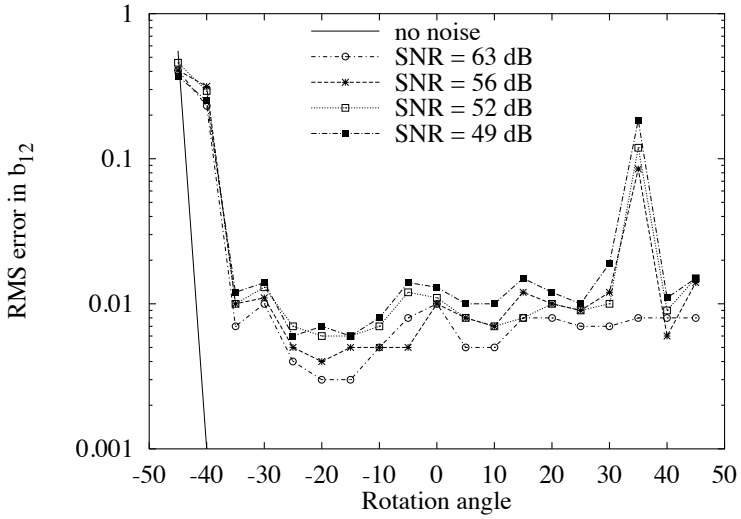



**Figure 6.5.** RMS error in $b_{12}$ versus rotation angle for the GauArea algorithm as a function of noise.

Figure 6.5 shows an example of an RMS error plot. Here, the RMS error in $b_{12}$ for algorithm GauArea is plotted against rotation angle (see Figure 6.2 for examples of what a rotated image looks like). The performance of GauArea is plotted for different amounts of noise. When there is no noise, the range is from -40 deg to 45 deg. As the noise is increased to 63 db the range drops to -35 deg to 45 deg. At higher noise levels, the plots show an interesting phenomenon. For noise $\geq 56$ dB, the RMS error increases dramatically

for a rotation angle of 35 degrees and then falls back with increasing rotation angle. A conservative evaluation requires that we assume that the algorithm is unstable once the error becomes large. Hence, for noise $\geq 56$ dB, the algorithm's range will be considered to be from -35 deg. to 30 deg and the algorithm will be regarded as unstable for angles greater than 30 degrees. This phenomenon (where the RMS error becomes really large and then falls) happens occasionally in some of the experiments. In this experiment, the spike at 35 degrees is caused by increasing the amount of noise. In other situations, the spike may also be caused by other factors (it may converge with a particular image but not with others). In all such situations, the range will be estimated in a conservative manner.

## 6.3 Discretizing Gaussians and Gaussian Derivatives

The derivations of the algorithms in the previous sections have assumed that the Gaussians and Gaussian derivatives were continuous functions. To apply them, they first need to be discretized. Errors arise due to the discretization process [23, 37]. A number of different procedures have been suggested in the literature. These include:

1. Block Averaging: Block averaging the continuous kernel over each pixel i.e. the filter value is integrated over each pixel and then sampled [23].

2. Discrete Derivative: The Gaussian is first sampled. The image is convolved first with the sampled Gaussian and the output convolved with a discrete version of the derivative. This approach is widely used.

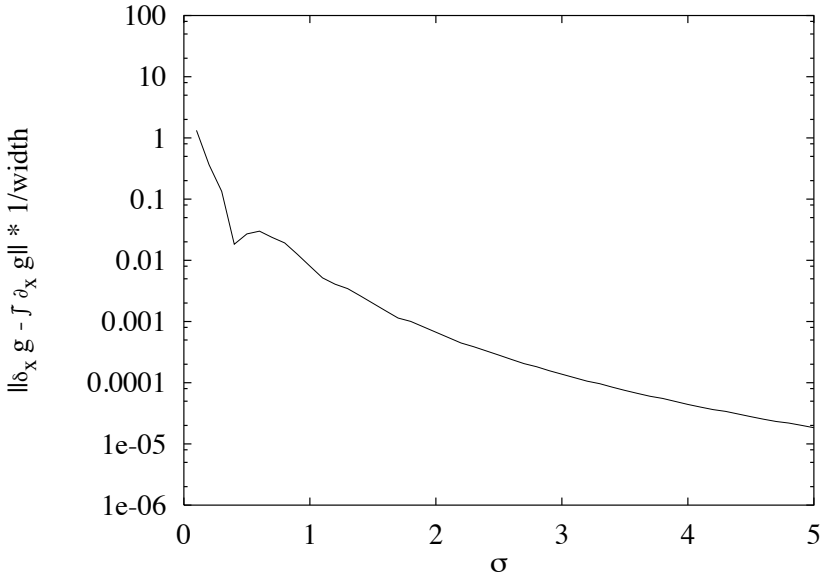
3. Sampled Gaussian Derivatives: The continuous version of the Gaussian or Gaussian derivative is directly sampled at each pixel and the sample values used for the discrete version of the filter.

4. Discrete Scale-Space: The values of the discrete Gaussian are computed using a discrete scale space. The image is filtered with the discrete Gaussian and then filtered with a discrete version of the derivative [37].

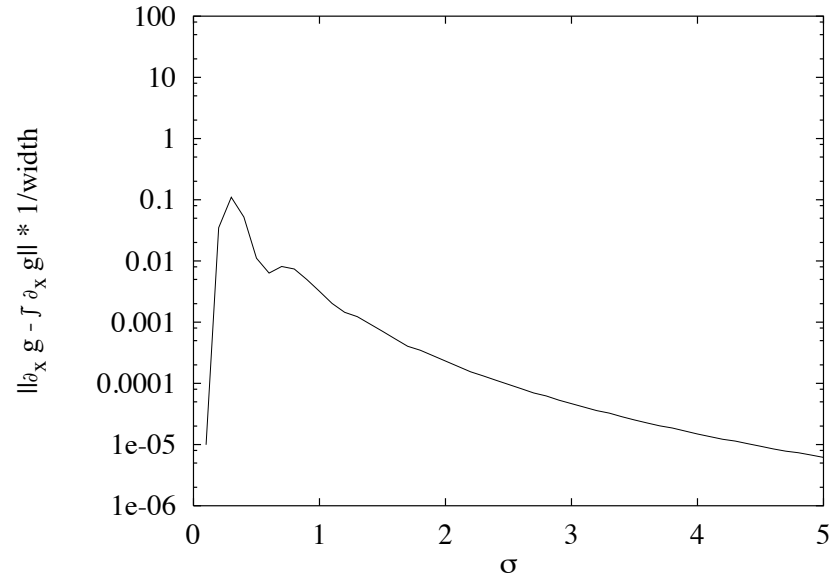
It may be argued that block averaging takes account of the imaging process and can, therefore, be assumed to be the best discretization [23, 37]. When a scene is imaged by a charge coupled device (CCD) camera, the output of each pixel is proportional to the total light falling over the entire area of each pixel [2]. It will be assumed here that block averaging is the best technique for discretizing Gaussian derivatives and other techniques will be compared with it.

Figure 6.6 compares derivative approximations for the first derivative of a Gaussian and Figure 6.7 for the second derivative of a Gaussian. Figure 6.6a compares discrete approximations to the first derivative of a Gaussian using discrete derivatives with the discretization obtained using block averaging, while Figure 6.6b compares first derivative approximations using sampled derivatives and block averaging. Similarly, Figure 6.7a compares the errors obtained using discrete derivatives instead of block averaging for approximating the second derivative of a Gaussian, while Figure 6.7b compares sampled derivatives with block averaging. In the figures, block averaging of the first derivative is denoted by $\int \partial_x g$, the discrete first derivative by $\delta_x g$ and the sampled first derivative by $\partial_x g$. A similar notation is used for the second derivative. The errors are computed by taking the difference of the respective quantities and than dividing by the filter width. The number of points used for computing the error is proportional to the filter width. Dividing by the filter width, therefore, eliminates this dependence. In the figure, all errors are plotted against the filter scale $\sigma$. For $\sigma > 0.4$, the sampled derivatives are a better approximation to block averaging than the discrete derivatives. As $\sigma$ increases, the errors fall quite rapidly especially for the sampled derivatives. Thus, for moderately large $\sigma$, the errors are sufficiently low that sampled derivatives may be used instead of block averaging. In this dissertation, sampled derivatives with $\sigma \geq 1.25$ will, therefore, be used.

[2]The area is actually better approximated as a weighted integral - the weight being a Gaussian [37].

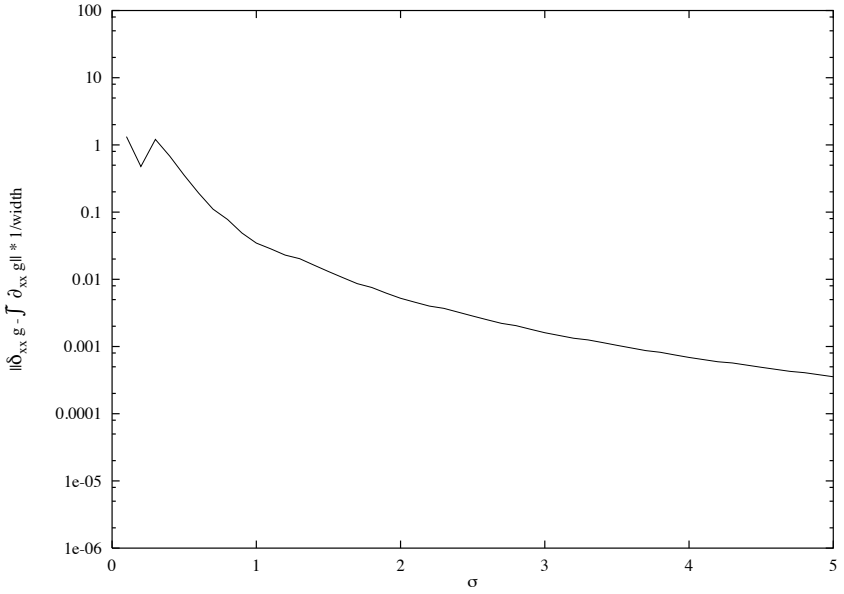



(a)

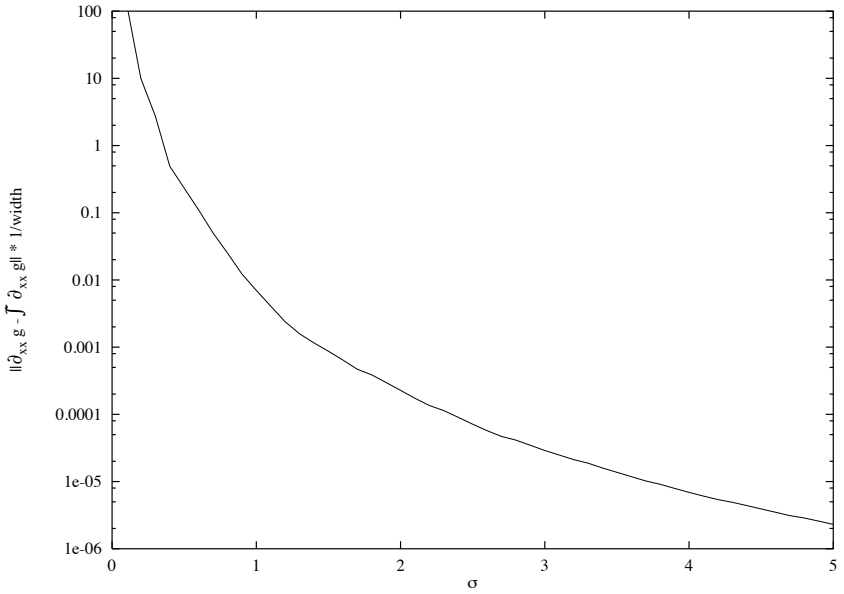



(b)

**Figure 6.6.** Comparison of different discretization approximations for the first derivative of a Gaussian. The top graph compares discrete derivatives with block averaging while the bottom graph compares sampled Gaussian derivatives with block averaging. Block averaging is denoted by $\int \partial_x g$, the discrete first derivative by $\delta_x g$ and the sampled first derivative by $\partial_x g$.




(a)




(b)

**Figure 6.7.** Comparison of different discretization approximations for the second derivative of a Gaussian. The top graph compares discrete derivatives with block averaging. The bottom graph compares sampled Gaussian derivatives with block averaging. Block averaging is denoted by $\int \partial_{xx} g$, the discrete derivative by $\delta_{xx} g$ and the sampled derivative by $\partial_{xx} g$.

Lindeberg derived a similar set of figures in [37] although he did not normalize them for filter width (Figures 5.9 and 5.10 in [37]). He also compared the discrete scale space with block averaging. The discrete scale space gives a poorer approximation to block averaging than the sampled derivative (see Lindeberg's figures). Thus, it will not be considered any further here.

## 6.4 Truncating Gaussians and Gaussian Derivatives

Gaussians and Gaussian derivatives are infinite in extent. However, most of their energies reside in a small region around the origin. Thus, for all practical purposes they can be truncated. Truncation also reduces the time taken to filter the images since the resulting kernel sizes are smaller. There has been some discussion about where Gaussians and Laplacian of Gaussians should be truncated [20, 23], but a general discussion of how Gaussian derivatives should be truncated seems to be absent in the literature.

Figure 6.8 shows the truncation errors for different values of the truncation radius. The truncation error is computed as follows:

$$TruncationError = \frac{\int_{-\infty}^{\infty} |G_{x^i}(x,\sigma)|dx - \int_{-k\sigma}^{k\sigma} |G_{x^i}(x,\sigma)|dx}{\int_{-\infty}^{\infty} |G_{x^i}(x,\sigma)|dx} \tag{6.6}$$

where $G_{x^i}(x,\sigma)$ is the ith order Gaussian derivatives (with $G$ denoting the Gaussian) and k is the truncation radius. In the above equation, the difference between the areas of the absolute values of the truncated function and the untruncated function is first computed. Then this difference is divided by the area of the absolute values of the untruncated filter to give the truncation error as a proportion of the area of the untruncated filter. Note that the truncation error is independent of $\sigma$. The truncation errors in Figure 6.8 were computed by summing over discrete versions of the filter using large $\sigma's$ instead of computing the integrals analytically (the difference should be insignificant).

As Figure 6.8 shows, for a given truncation radius, the error increases with the order of the derivative. Many researchers assume that it suffices to truncate Gaussians such that the

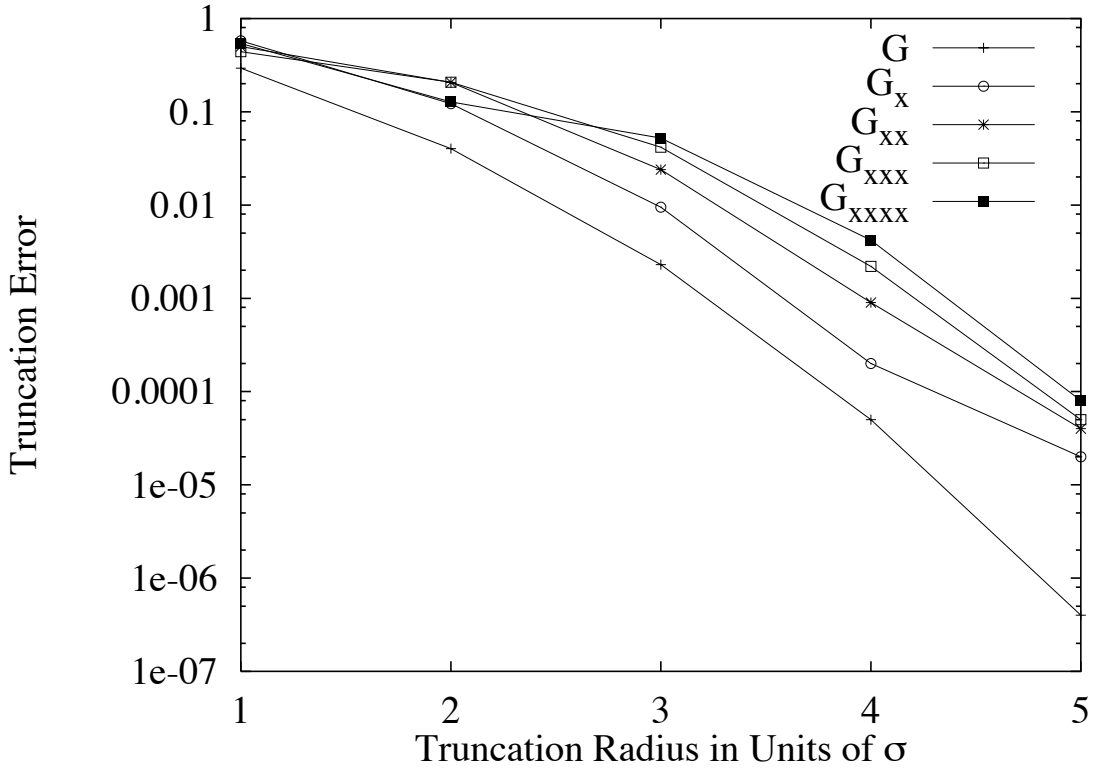



**Figure 6.8.** Truncation errors as a function of the truncation radius for Gaussians and Gaussian derivative filters. G denotes the Gaussian and $G_x$, $G_{xx}$, $G_{xxx}$ and $G_{xxxx}$ denoting the first, second, third and fourth Gaussian derivatives respectively. The truncation errors are taken as a proportion of the total area of the absolute value of filter.

truncation radius is $\pm 2\sigma$ i.e. the filter width = $4\sigma$. For a truncation radius of $2\sigma$, $96\%$ of the energy of the Gaussian is contained within the filter width (i.e. the truncation error is 0.04). But for the same truncation radius, Gaussian derivatives have a much larger truncation error. For example, the first derivative of the Gaussian has an error of about 12 % if it is truncated to within $\pm 2\sigma$ while higher derivatives have a much larger error. The question arises as to what level of error is significant.

We shall argue here that a truncation radius of $4\sigma$ should be used for Gaussian derivatives up to order 4. As the graph (Figure 6.8) shows that at $4\sigma$, the truncation error is less than 0.5 % even for the 4th derivative. The following graphs show that a truncation ra-

dius of $2\sigma$ or even $3\sigma$ causes not only quantitative errors but large qualitative errors in the appearance of the filtered signal.

The top row of Figure 6.9 shows a 1D intensity function. The other rows show successively the first, second, third and fourth derivatives of Gaussians of the intensity function on the top row. The Gaussian derivatives are computed for $\sigma = 10$ and a truncation radius of $\pm 4\sigma$ (similar figures are produced for other values of $\sigma$). Qualitatively, the $(n+1)^{th}$ derivative has zero crossings wherever the $n^{th}$ derivative has a maximum or minimum. For example, the first derivative (second row) has zero crossings wherever the intensity function (zeroth row) has a maximum or minimum. The signals in this graph are examples of the kind of results desired.

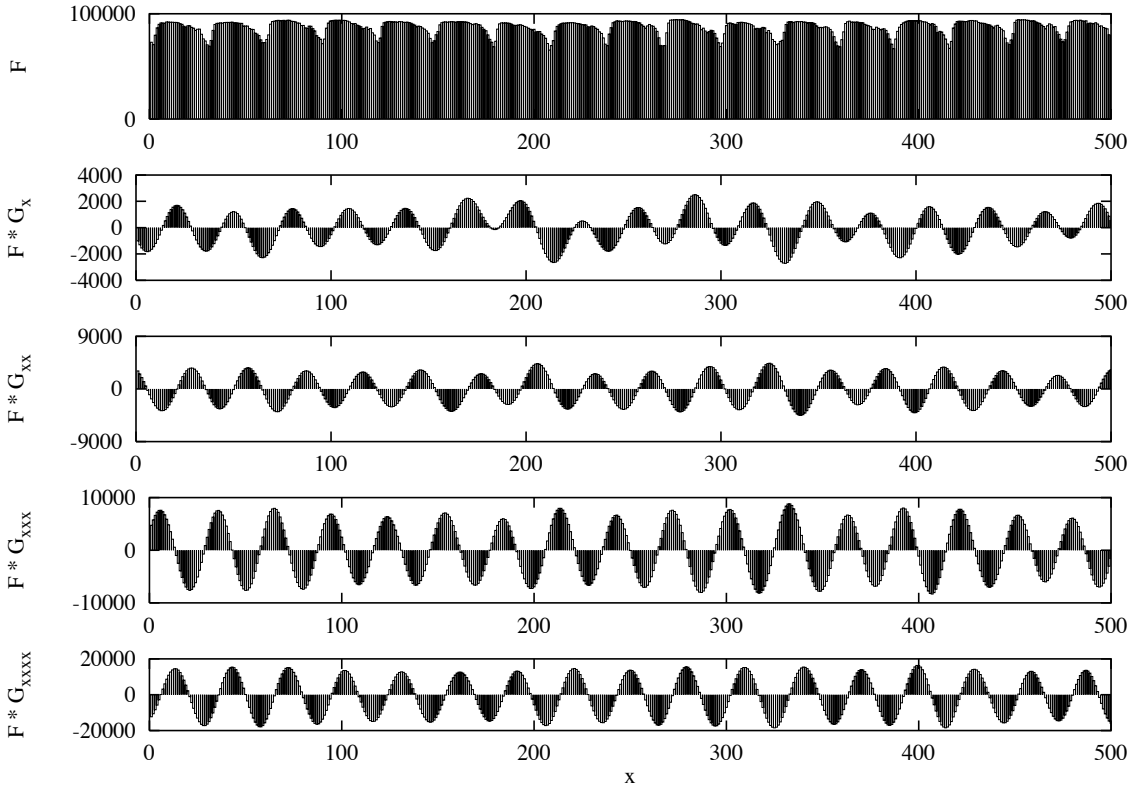



**Figure 6.9.** Plots of Gaussian derivatives computed for a truncation radius of $\pm 4\sigma$

Figure 6.10 shows the same plots as for 6.9 but plotted for a smaller truncation radius of $\pm 2\sigma$. The second and fourth derivatives are computed incorrectly and are different from

those in Figure 6.9. Wherever the first derivative has an extremum, the second derivative should have a zero crossing. Similarly, wherever the third derivative has a maxima or minima, a zero crossing should occur in the fourth derivative. In Figure 6.10, however the second and fourth Gaussian derivatives are always negative and hence computed incorrectly. Although the first derivative appears qualitatively correct (has the same zero crossings as before), it does show quantitative differences. The peaks in Figure 6.10 are larger than those in Figure 6.9. It is clear that a truncation radius of $\pm 2\sigma$ gives poor results.

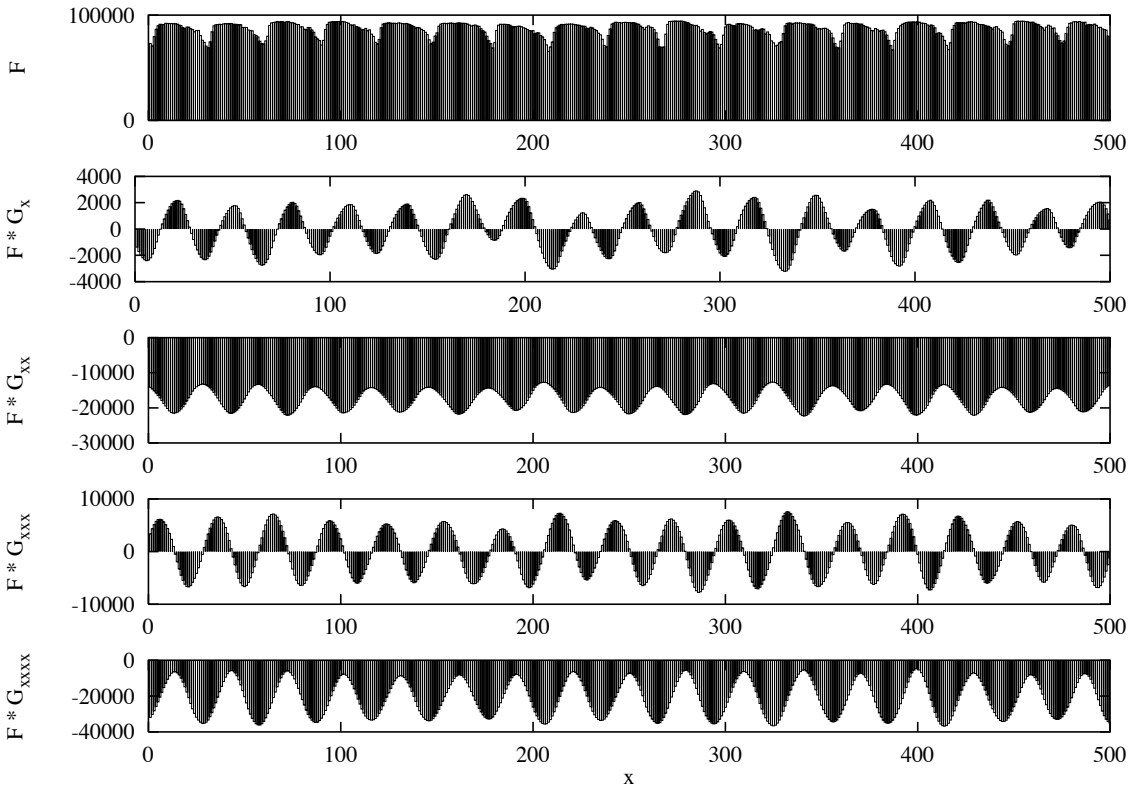



**Figure 6.10.** Plots of Gaussian derivatives computed for a truncation radius of $\pm 2\sigma$

Figure 6.11 shows the same plots as for 6.9 but for a truncation radius of $\pm 3\sigma$. There is some improvement in the computation of the zero crossings of the second and fourth Gaussian derivatives. Notice that now the second and fourth derivatives have zero crossings wherever the first and third derivatives respectively show maxima and minima. However, the positive portions of the second and fourth derivatives are small compared to the negative

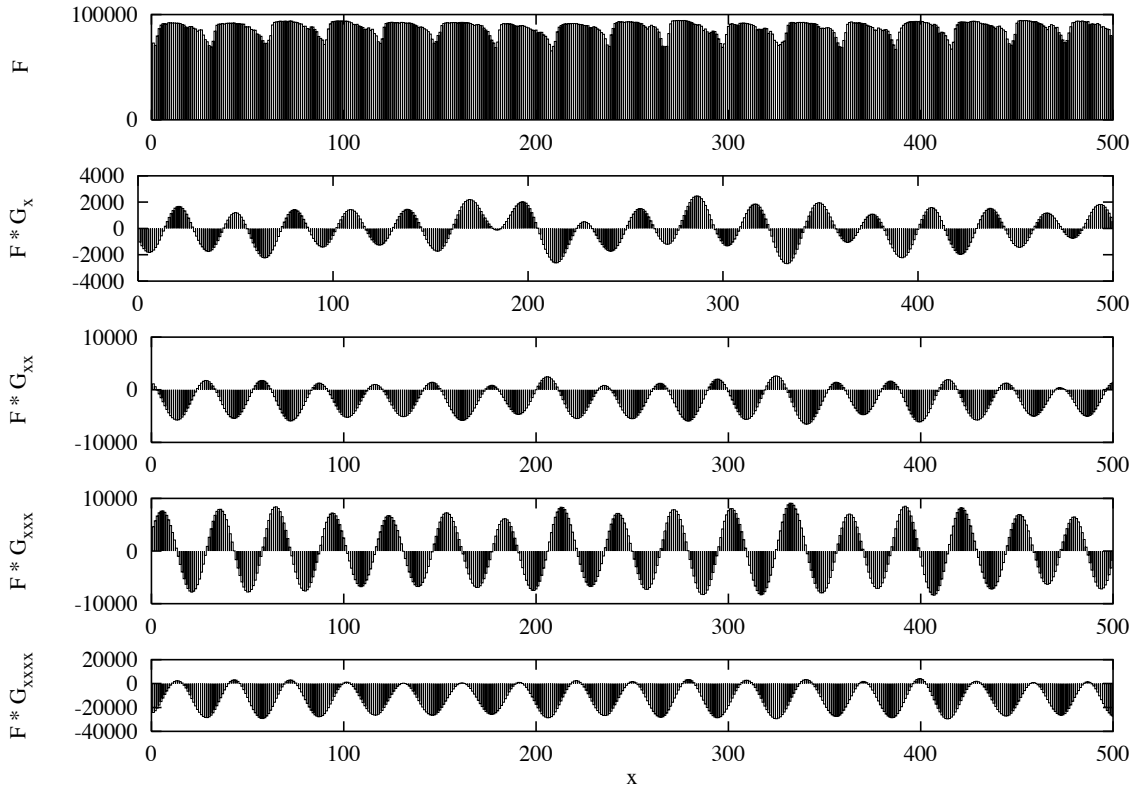



**Figure 6.11.** Plots of Gaussian derivatives computed for a truncation radius of $\pm 3\sigma$

portions (and small compared to the correct answers). Thus, a truncation radius of $\pm 3\sigma$ causes errors in the computation of the derivatives.

As a further example, the effect of truncation errors is demonstrated on the algorithm GauArea. The performance of the algorithm is plotted (Figure 6.12) for different filter widths $4\sigma$, $6\sigma$ and $8\sigma$ (i.e. truncation radius of $\pm 2\sigma, \pm 3\sigma$ and $\pm 4\sigma$ respectively). The RMS error for $b_{11}$ is plotted, RMS errors for the other parameters are shown in Figures C.1(a)-C.1(f) in Appendix C. The experiment is performed on two random dot images, one of which is scaled by different amounts with respect to the other. (see 6.2.1.1). It is clear that at width $4\sigma$, the algorithm has less than half the range at the other widths. At width $8\sigma$ gives a slightly larger range than at radius $6\sigma$ (although it is surprising that the errors are not larger). Similar results were obtained with other affine transformations.

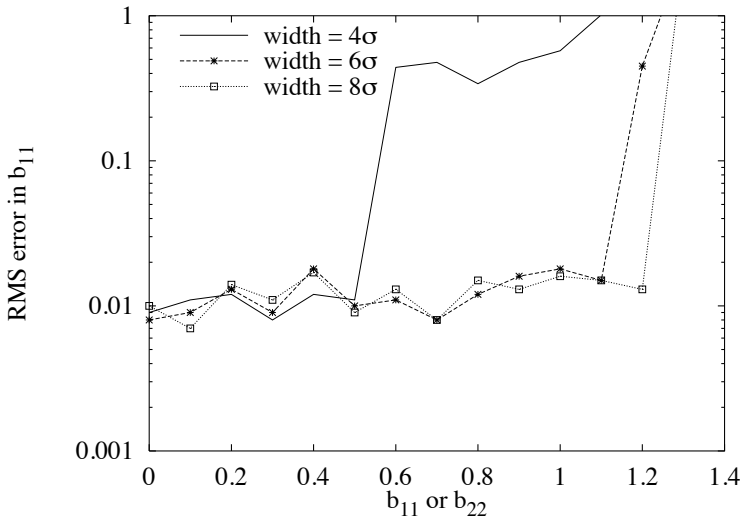



**Figure 6.12.** RMS error in $b_{11}$ versus $b_{11}$ for the GauArea algorithm as a function of filter widths.

In this dissertation, Gaussian derivatives of order 3 or less are computed. Thus all further experiments will be carried out with a truncation radius of $\pm 4\sigma$ (filter width of $8\sigma$).

## 6.5 Effect of Varying the Filter Scale

The following experiments test the effects of varying the filter scales on the algorithms. The algorithms used Gaussian and Gaussian derivative filters at two different scales, differing by a factor of $\sqrt{2}$ For example, one set of scales used was $\sigma = 1.25$ and $\sigma = 1.25 \times \sqrt{2}$. In all four different sets of filter scales were used. They were

1. $\sigma = 1.25$ and $\sigma = 1.25 \times \sqrt{2} = 1.768$
2. $\sigma = 1.768$ and $\sigma = 1.768 \times \sqrt{2} = 2.5$
3. $\sigma = 2.5$ and $\sigma = 2.5 \times \sqrt{2} = 3.54$
4. $\sigma = 3.54$ and $\sigma = 3.54 \times \sqrt{2} = 5.0$

The scale sets will be referred to by the base (lowest) scales used. For example, the first scale set will be referred to as scale set $\sigma = 1.25$. Three sets of experiments were con-

ducted; the first experiment varies the scaling (see Section 6.2.1.1) of the images, the second the rotation (see Section 6.2.1.2) and the third the shear (see Section 6.2.1.3). Graphs are plotted for four different sets of scales for the algorithms GauArea and DGauArea.

### 6.5.1 GauArea: Performance as a Function of Filter Scale

Figures 6.13-6.15 show the performance of GauArea (see 5.4.1) for scalings, rotations and shears (see Figures 6.1-6.3 for examples of these transformations) respectively as the filter scale is varied. In two of the figures, the RMS error in the affine parameter $b_{11}$ is plotted against scale change (i.e $b_{11}, b_{22}$), and shear deformation respectively while the parameter $b_{12}$ is plotted against rotation (see Figure 6.14) since it shows larger errors than $b_{11}$ in this case. The other affine parameters show similar trends. For completeness, the plots for all the affine parameters are shown in Figures C.2(a)-C.4(f) in Appendix C. With random dot images the algorithm GauArea shows an increase in error with scale for all transformations. This may be attributed to the poorer accuracy in localization as the scale is increased.

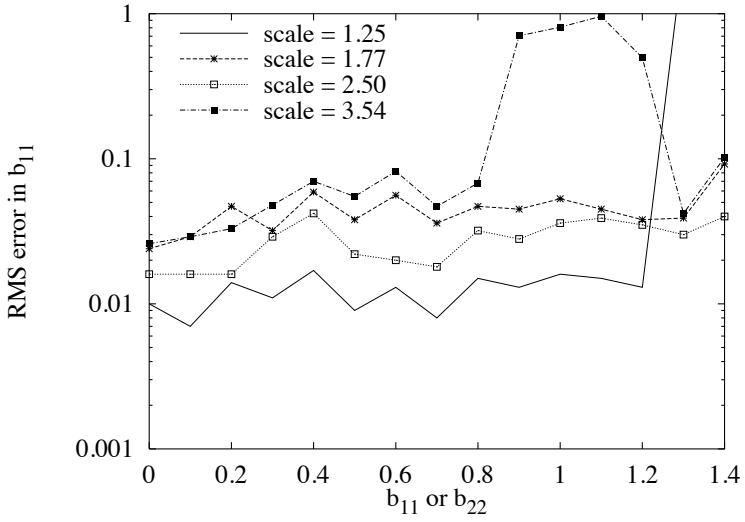



**Figure 6.13.** RMS error in $b_{11}$ versus scale change ($b_{11}$) for the GauArea algorithm as a function of filter scale.

There is also a trend towards a decrease in range with increasing filter scale for scalings and shears, while for rotations the trend is towards an increase in range with scale. For image scalings (Figure 6.13) at scale 1.25 the range is $0 \leq b_{11} \leq 1.2$. It increases slightly for filter scales 1.768 and 2.50 to $0 \leq b_{11} \leq 1.4$. For filter scale 3.54 the range falls to $0 \leq b_{11} \leq 0.8$. Notice that at scale 3.54, the error rises dramatically and then drops. As discussed before (Section 6.2.2), the algorithm is interpreted as being unstable beyond $b_{11} = 0.8$. We use the criterion that the algorithm's useful range occurs when the RMS error is less than 0.1 (see Section 6.2.2). Clearly with some other criterion the range would be much larger.

For rotations (Figure 6.14), the range is from -35 deg. to 30 deg. at the smallest scale (1.25). It increases to -45 deg. to 45 deg. for the intermediate scales before dropping to -35 deg. to 45 deg. for the largest scale (3.54).

The range of shear deformations handled by GauArea (Figure 6.15) is (0-0.48) for the smallest scale (1.25), it decreases to (0-0.40) for the next scale (1.768) before dropping to (0-0.32) for the two largest scales.

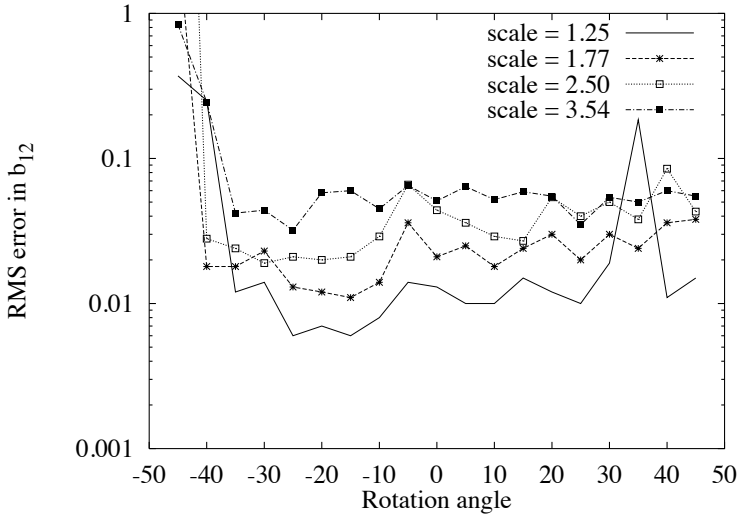



**Figure 6.14.** RMS error in $b_{11}$ versus rotation angle for the GauArea algorithm as a function of filter scale.

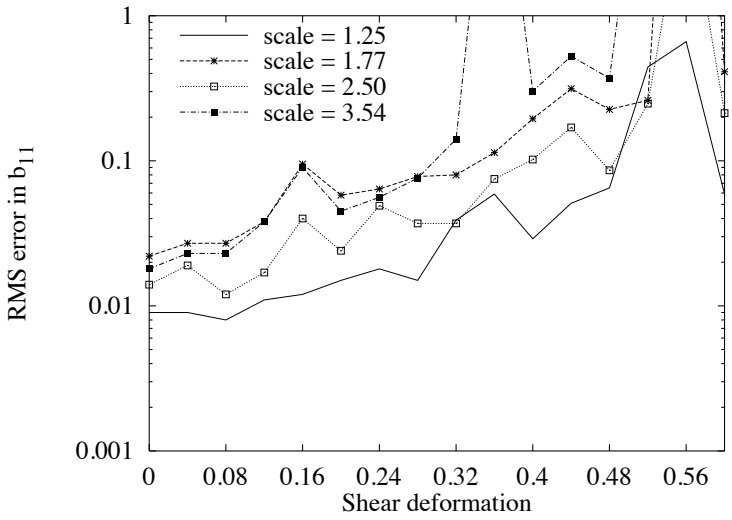



**Figure 6.15.** RMS error in $b_{11}$ versus shear deformation for the GauArea algorithm as a function of filter scale.

### 6.5.2 DGauArea: Performance as a Function of Filter Scale

The experiments are repeated with algorithm DGauArea (i.e. the using the first derivatives of the Gaussian). The RMS errors for $b_{11}$ are displayed in Figures 6.16-6.18 and the errors for all the affine parameters are displayed in Figures C.5(a)-C.7(a) in Appendix C.

For scalings (Figure 6.16), DGauArea has a much lower error than GauArea (less than half the error). However, for the lower scales the range of DGauArea is much smaller than GauArea (the range is 0-0.5). For the largest scale (3.54) on the other hand, the range for DGauArea is (0-1) and is larger than for GauArea. It will turn out later (see Section 6.11 that for real images, the ability to use large scales is useful and, therefore, DGauArea will in general be a more useful algorithm than GauArea.

For rotations (Figure 6.17), DGauArea shows a trend for range to increase with filter scale. At scale 1.25, the range is $(-25\,\mathrm{deg}, 25\,\mathrm{deg})$, it increases to $(-35\,\mathrm{deg}, 45\,\mathrm{deg})$ for scale 1.768 and further increases to $(-40\,\mathrm{deg}, 45\,\mathrm{deg})$ for scale 2.50. At the largest scale the range is $(-45\,\mathrm{deg}, 40\,\mathrm{deg})$.

DGauArea handles shear deformations (Figure 6.17) from 0 to 0.48 for the first three scales and from 0 to 0.32 for the largest scale.

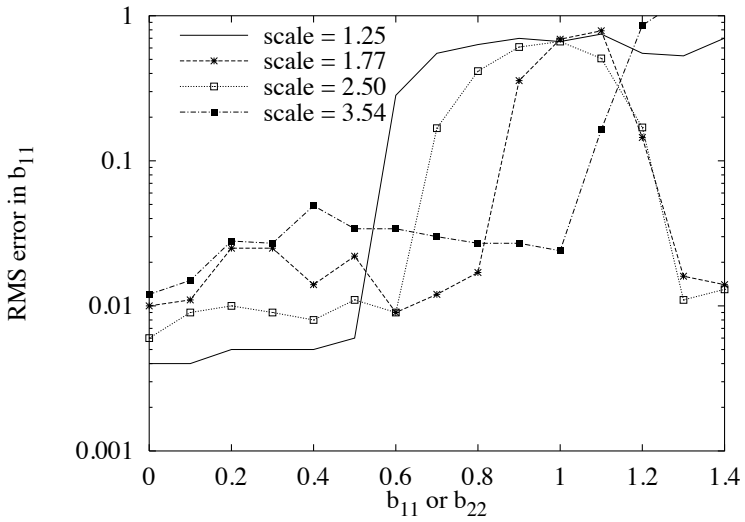



**Figure 6.16.** RMS error in the $b_{11}$ versus scale change ($b_{11}$) for the DGauArea algorithm as a function of filter scale.

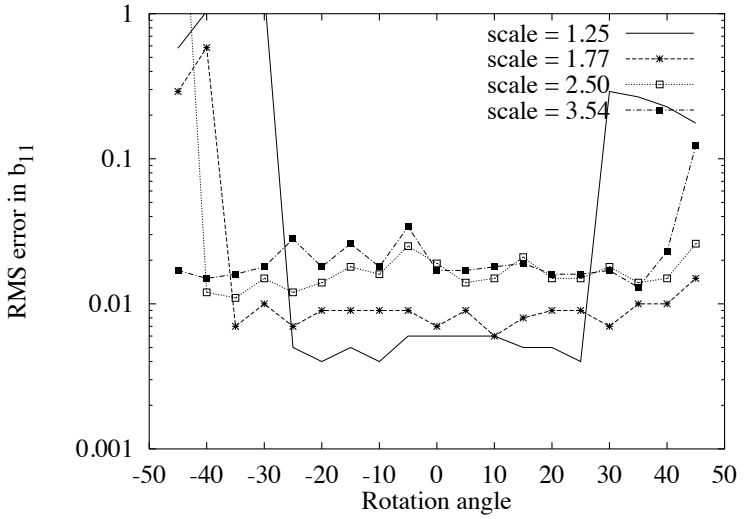



**Figure 6.17.** RMS error in $b_{11}$ versus rotation angle for the DGauArea algorithm as a function of filter scale.

### 6.5.3 Comments on Varying the Filter Scale

As the scale of the filters is increased, their range increases. This is true for deformations which are composed of scale changes and rotations. For shear deformations, the filter range is actually smaller for the largest scale set used. The increase in range due to larger scales may be attributed to the use of larger support regions. The accuracy of the

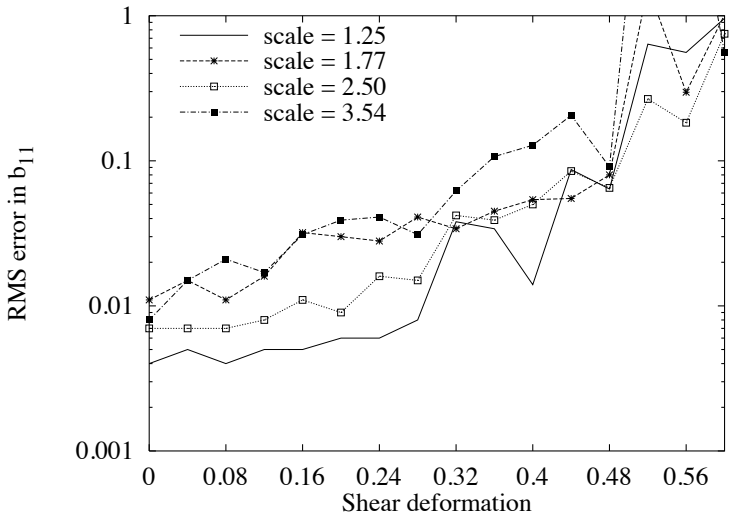



**Figure 6.18.** RMS error in $b_{11}$ versus shear deformation for the DGauArea algorithm as a function of filter scale.

algorithms, however, goes down with scale and maybe attributed to the poorer localization (i.e. translation estimates) obtained at higher scales.

Overall DGauArea is more accurate than GauArea for rotations and shears. For scalings, DGauArea has a much lower range than GauArea for the smaller scales while for the largest scale it has a higher range. The RMS error of DGauArea is in general much lower than for GauArea.

## 6.6 Effect of Varying the Number of Filters Used

The following experiments will test the effect of varying the number of filters on the algorithms. The following three sets of filters were tried.

1. Just one filter at $\sigma = 1.25$.

2. Two filters, one at $\sigma = 1.25$ and one at $\sigma = 1.25 \times \sqrt{2} = 1.768$

3. Three filters, the first at $\sigma = 1.25$, the second at $\sigma = 1.768$ and the third at $\sigma = 1.25 \times 2 = 2.0$

In the following experiments, a window of size 13 by 13 was used and every point within this window was used. The filter widths were set to $8\sigma$. As before three sets of experiments will be conducted; the first experiment varies the scaling of the images, the second the rotation and the third the shear. Graphs are plotted for the three different sets of filter scales for both algorithms GauArea and DGauArea.

### 6.6.1 GauArea: Performance as a Function of Number of Filters Used

The performance of GauArea as a function of the number of filters used is shown in Figures 6.19-6.21 for scalings, rotations and shears (see Figures 6.1-6.3 for examples of these transformations). It is clear from the figures that the accuracy of the algorithm remains the same even with additional filters. The range of the algorithm increases slightly (for scalings and rotations) with the number of filters used. For completeness, the plots for all the affine parameters are shown in Figures C.8(a)-C.10(f) in Appendix C.

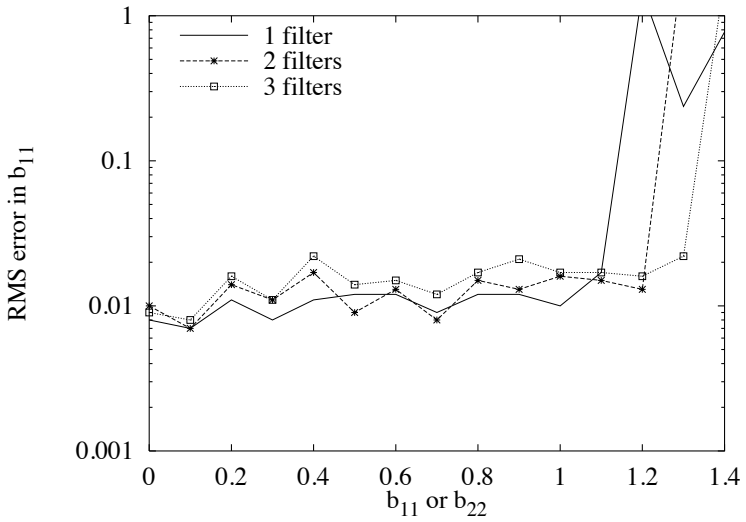



**Figure 6.19.** RMS error in $b_{11}$ versus scale change ($b_{11}$) for the GauArea algorithm as a function of the number of filters used.

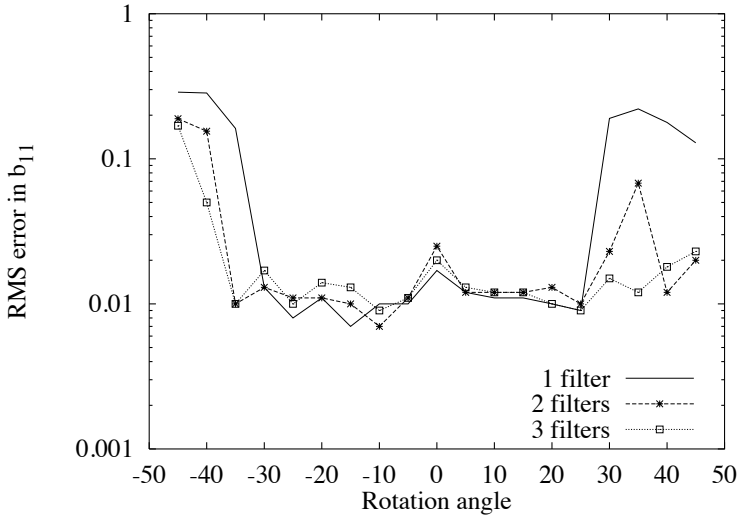



**Figure 6.20.** RMS error in $b_{11}$ versus rotation angle for the GauArea algorithm as a function of the number of filters used.

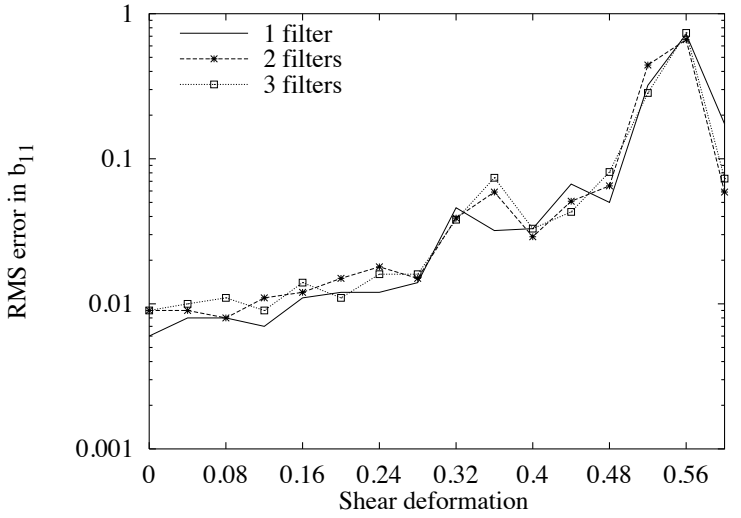



**Figure 6.21.** RMS error in $b_{11}$ versus shear deformation for the GauArea algorithm as a function of the number of filters used.

### 6.6.2 DGauArea: Performance as a Function of Number of Filters Used

The effect of varying the number of filters on the performance of DGauArea is shown in Figures 6.22-6.24 for scalings, rotations and shears (see Figures 6.1-6.3 for examples of these transformations). The figures show that there is a slight advantage to using at least

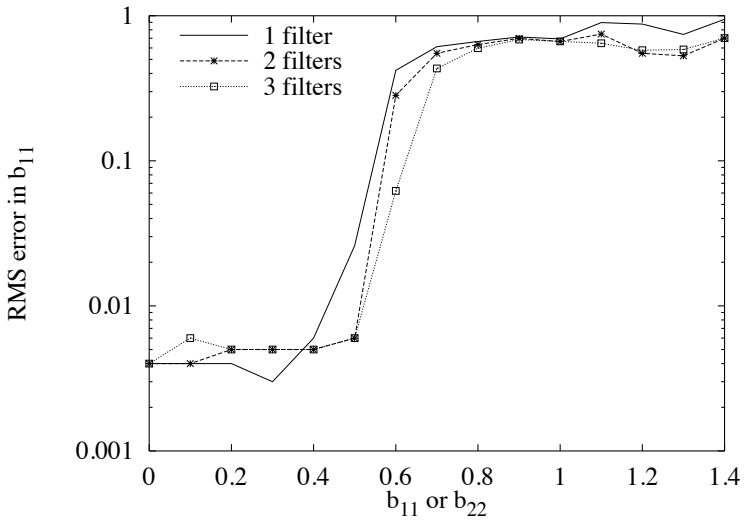



**Figure 6.22.** RMS error in $b_{11}$ versus scale change ($b_{11}$) for the DGauArea algorithm as a function of the number of filters used.

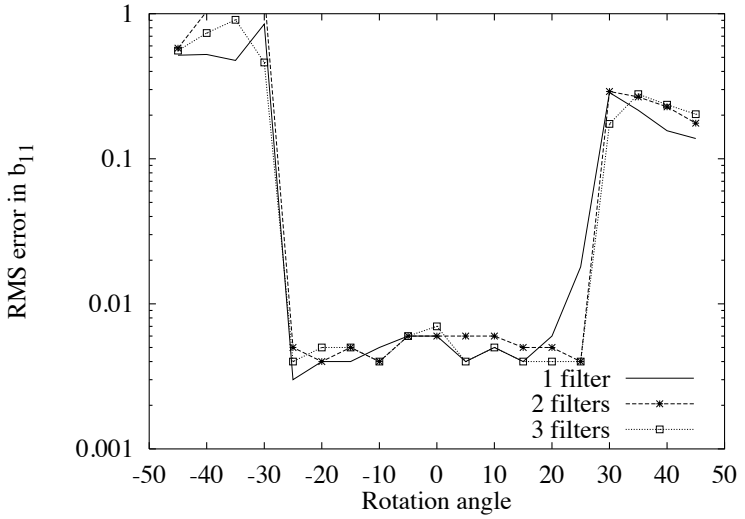



**Figure 6.23.** RMS error in $b_{11}$ versus rotation angle for the DGauArea algorithm as a function of number of scales used simultaneously.

2 filters but there is not much advantage to using more than 2 filters. As before algorithm DGauArea is much more accurate than GauArea within its effective range.

For completeness, the plots for all the affine parameters are shown in Figures C.11(a)-C.13(f) in Appendix C.

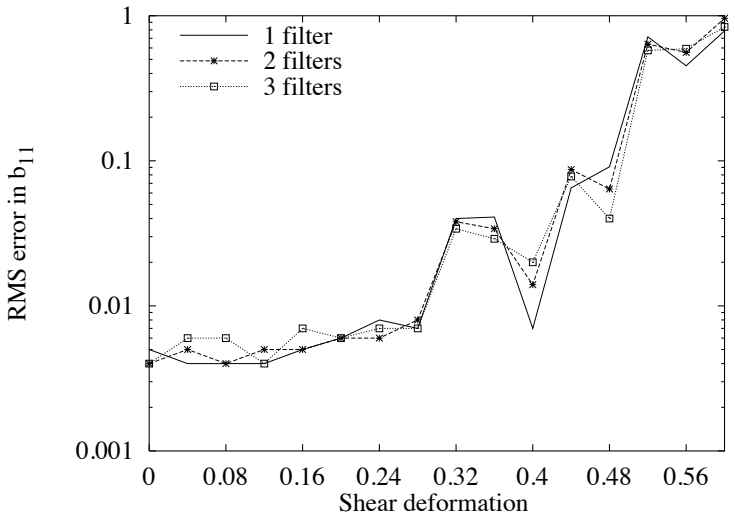



**Figure 6.24.** RMS error in $b_{11}$ versus shear deformation for the DGauArea algorithm as a function of number of scales used simultaneously.

## 6.7 Effect of Window Size

The effect of changing the window size on the performance of the algorithms is discussed in this section. As the experiments show using many more points for computing the affine transform usually gives more accuracy. However, for some kinds of transforms - principally rotations - larger window sizes reduce the range. As before three sets of experiments will be conducted; the first experiment varies the scaling of the images, the second the rotation and the third the shear (see Figures 6.1-6.3 for examples of these transformations). Graphs are plotted for three different window sizes 9x9, 13x13 and 17x17.

In the following experiments, unless otherwise stated, filters at two scales were used $\sigma = 1.25$ and $\sigma = 1.25 \times \sqrt{(2)} = 1.768$. The filter widths were set to $8\sigma$.

### 6.7.1 GauArea: Effect of Window Size

Figures 6.25-6.27 show the effect of changing window size on algorithm GauArea. The figures show graphs for three different window sizes 9x9, 13x13 and 17x17.

For scalings (Figures 6.25), it is clear that increasing the window size improves the accuracy of GauArea. Thus for example, the 17x17 window measures affine transforms

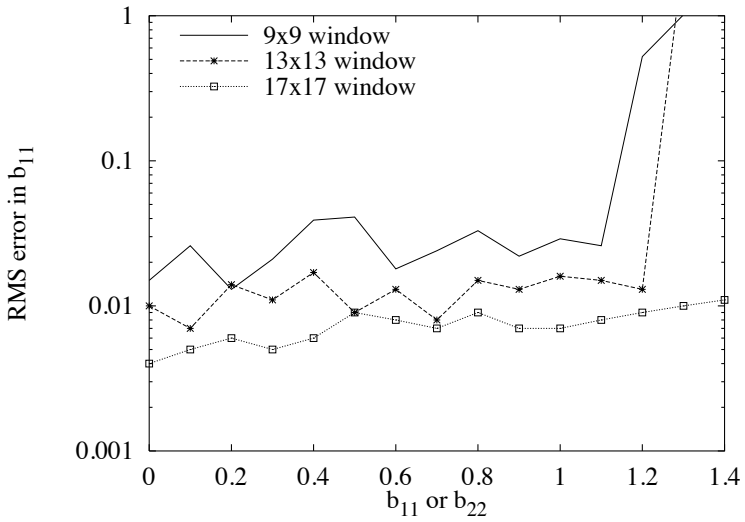



**Figure 6.25.** RMS error in $b_{11}$ versus scale change ($b_{11}$) for the GauArea algorithm as a function of window size.

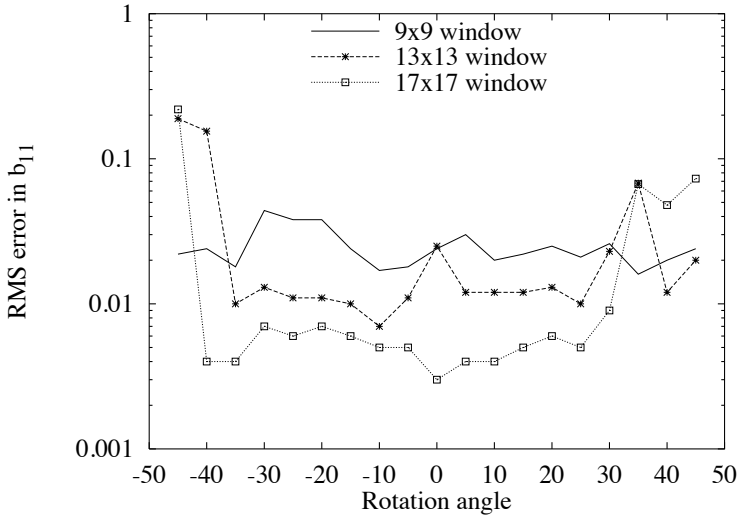



**Figure 6.26.** RMS error in $b_{11}$ versus rotation angle for the GauArea algorithm as a function of window size.

almost twice as accurately as the 9x9 window. The increase in accuracy is due both to the larger number of points used and the larger region used to estimate the affine transform. Increasing the window size also increases the range of the algorithms.

For rotations (Figures 6.26), increasing window size also increases the accuracy. However, the range falls with increasing window size. The decrease in range may be understood

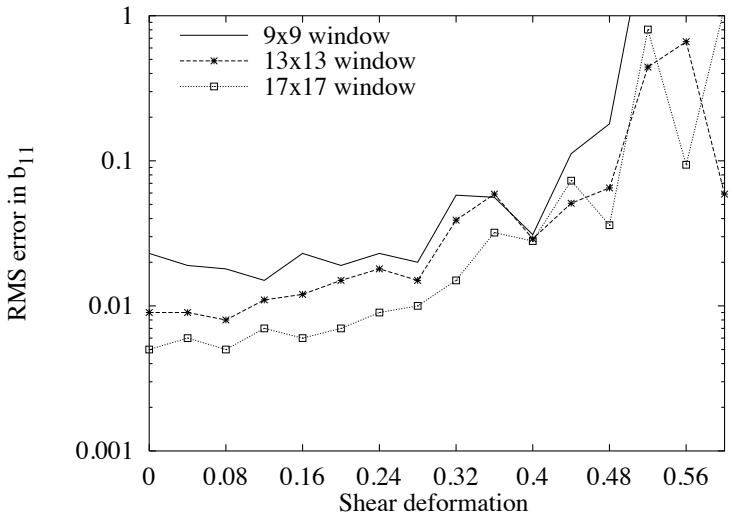



**Figure 6.27.** RMS error in $b_{11}$ versus shear deformation for the GauArea algorithm as a function of window size.

as follows: Since the algorithm is iterative, it is essential that the initial estimate be not far from the correct answer. As the rotations increase the corresponding points in the second image are further away and this distance increases with window size. This leads to the decrease in range with increasing window size.

For shears (Figures 6.27), again there is an increase in accuracy (as expected). When the affine transform is small, the error at a window size of 17 by 17 is 1/4 that at a window size of 9 by 9. For larger affine transforms a window size 17 by 17 has an error rate 1/2 that at a window size of 9 by 9. There is also a slight increase in range with window size.

As before, plots for all the affine parameters are shown in Figures C.14(a)-C.16(f) in Appendix C.

### 6.7.2 DGauArea: Effect of Window Size

For scalings, Figure 6.28 shows the effect of window size on DGauArea. Appendix C contains a complete set of plots (Figures C.17(a)-C.17(f))for all affine parameters. As for GauArea, the accuracy of DGauArea increases with window size. The range of DGauArea seems to show anomalous behaviour - the range is the same at window sizes of 9 by 9 and

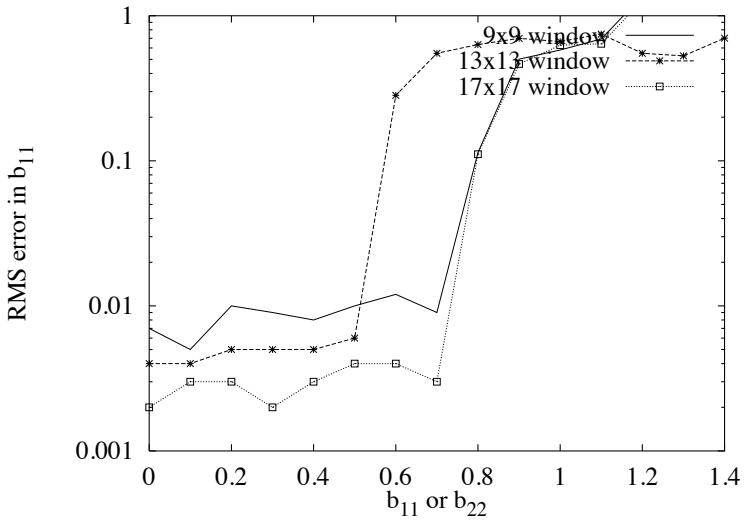



**Figure 6.28.** RMS error in $b_{11}$ versus scale change ($b_{11}$) for the DGauArea algorithm as a function of window size.

17 by 17 but is smaller at an intermediate range. This anamoly is present even if the noise is removed and is probably an artifact of the particular image used.

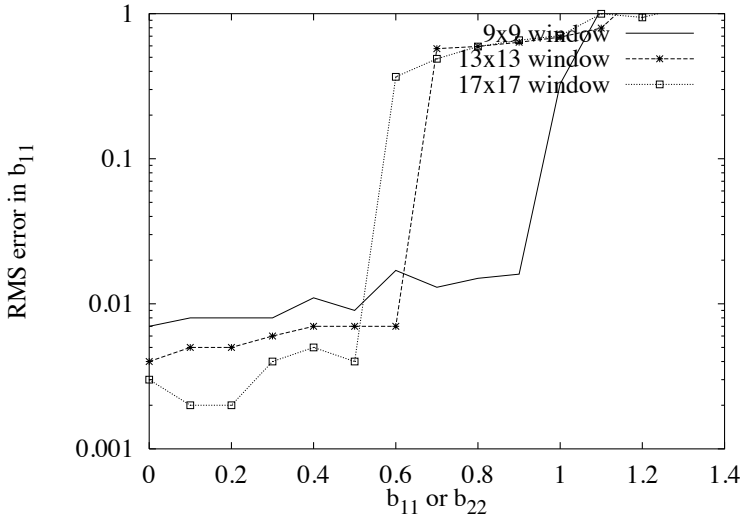



**Figure 6.29.** RMS error in $b_{11}$ versus scale change ($b_{11}$) for the DGauArea algorithm as a function of window size at filter scale 1.25. A different random dot image is used as compared to Figure 6.28

.

To clarify the range behaviour of DGauArea further experiments were conducted using a different random dot image (generated with a different seed). Figure 6.29 shows the performance of DGauArea on a different image but with the same parameters as before. With increasing window size, a clear decrease in range occurs.

DGauArea's behaviour is ,however, different when much larger filter scales are used. Figure 6.30 shows the performance of DGauArea as a function of window size for filter scale 2.5. It is seen that the range now increases with window size ( a reversal of the previous behaviour). As we have noted before (Section 6.5), the performance of DGauArea is better at larger scales and the experiments here confirm the trend.

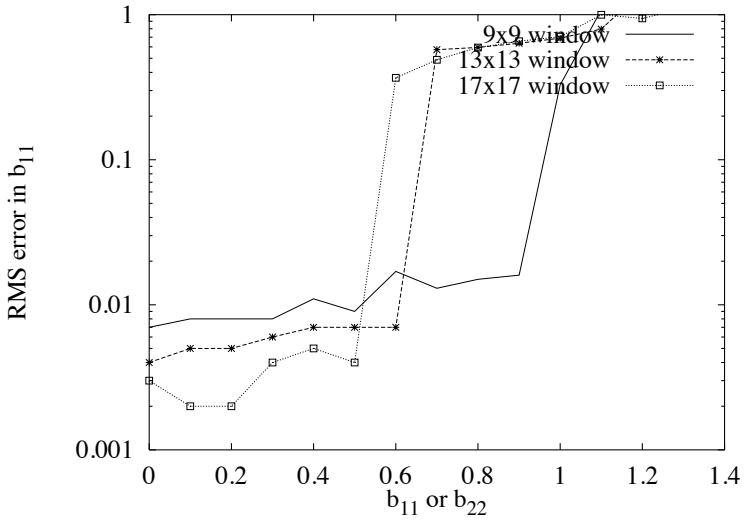



**Figure 6.30.** RMS error in $b_{11}$ versus scale change ($b_{11}$) for the DGauArea algorithm as a function of window size at filter scale 2.5.

For rotations (Figures 6.31), DGauArea behaves the same way as GauArea. The accuracy improves with window size and the range decreases with window size. For the same rotation angle and the same window size, DGauArea is roughly twice as accurate as GauArea. As before, a complete set of plots (Figures C.18(a)-C.18(f)) for all affine parameters is availabe in Appendix C.

For shears (Figures 6.32), again there is an increase in accuracy (as expected). The error at a window size of 17 by 17 is 1/4 that at a window size of 9 by 9. The range is more

or less the same for all window sizes. Appendix C contains plots (Figures C.19(a)-C.19(f)) for all the affine parameters.

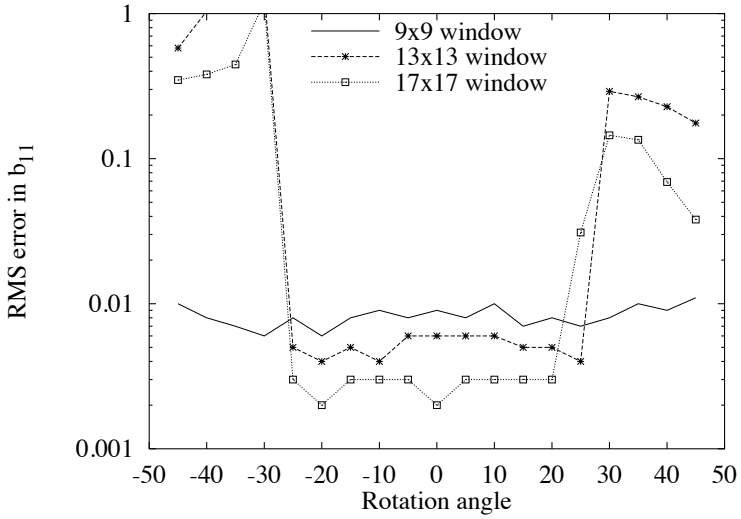



**Figure 6.31.** RMS error in $b_{11}$ versus rotation angle for the DGauArea algorithm as a function of window size.

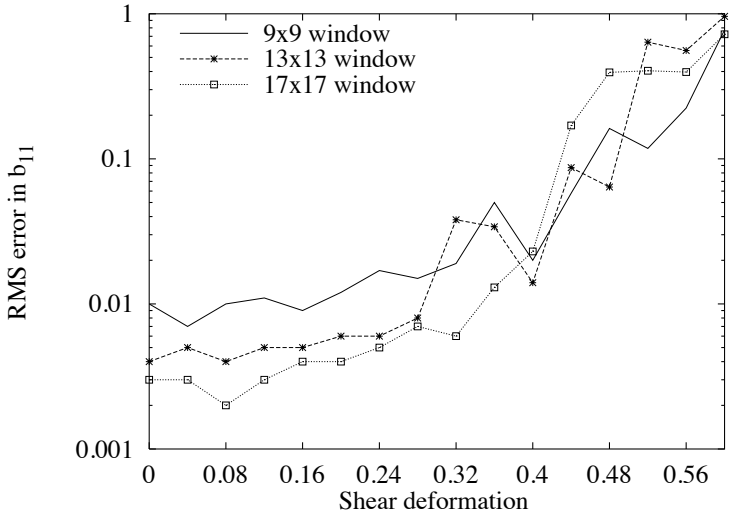



**Figure 6.32.** RMS error in $b_{11}$ versus shear deformation for the DGauArea algorithm as a function of window size.

## 6.8 Noise Performance

The performance of GauArea and DGauArea with different levels of noise added is investigated in this section. Experiments were done with scalings, rotations and shears (see Figures 6.1-6.3 for examples of these transformations). Different amounts of noise were added to the images as described in Section 6.2.1. The SNR was computed as in 6.1.

In the following experiments, filters at two scales were used $\sigma = 1.25$ and $\sigma = 1.25 \times \sqrt{(2)} = 1.768$. The filter widths were set to $8\sigma$.

### 6.8.1 GauArea: Noise Performance

Figures 6.33-6.35 show the effect of noise on the algorithm GauArea.

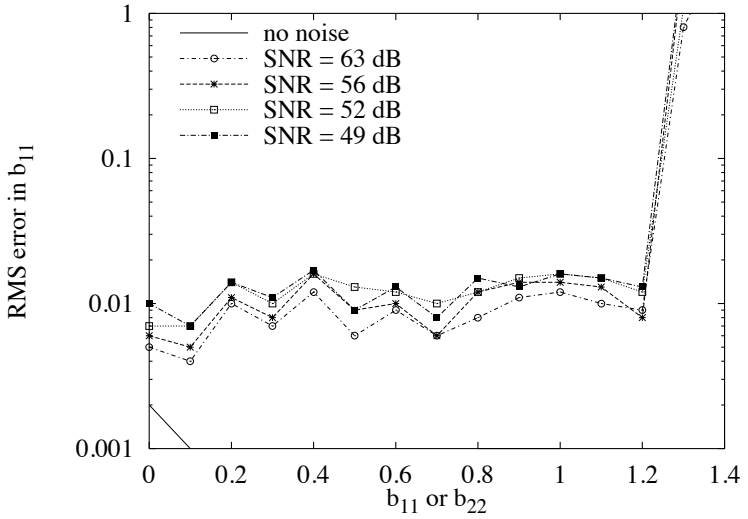



**Figure 6.33.** RMS error in $b_{11}$ versus scale change ($b_{11}$) for the GauArea algorithm as a function of noise.

For scalings (Figure 6.33) without any noise, GauArea's range is large ($0 \leq b_{11} \leq 1.4$) and the RMS error is less than 0.001 for the most part. Adding noise decreases the range to 1.2 and increases the error substantially. Larger amounts of noise cause the error rate to increase. At 49 dB, the noise is roughly double that at 63 dB.

For rotations (Figure 6.34) without any noise, GauArea's range is from -40 deg. to 45 deg and the RMS error is again less than 0.001 within the effective range. For higher noise

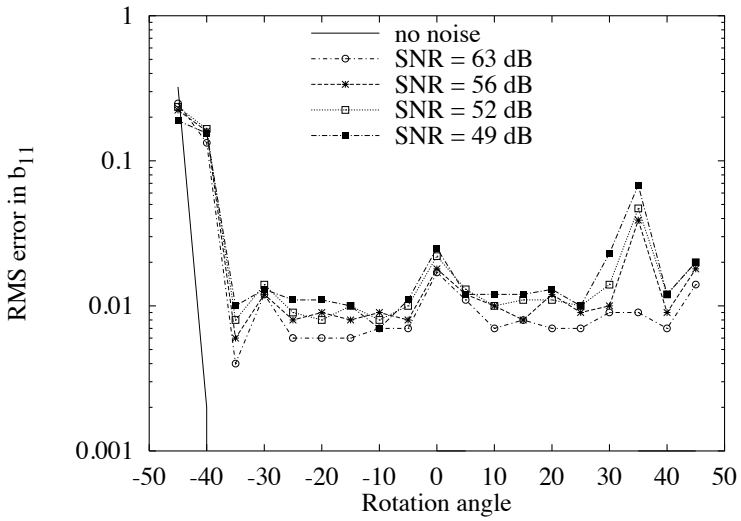



**Figure 6.34.** RMS error in $b_{12}$ versus rotation angle for the GauArea algorithm as a function of noise.

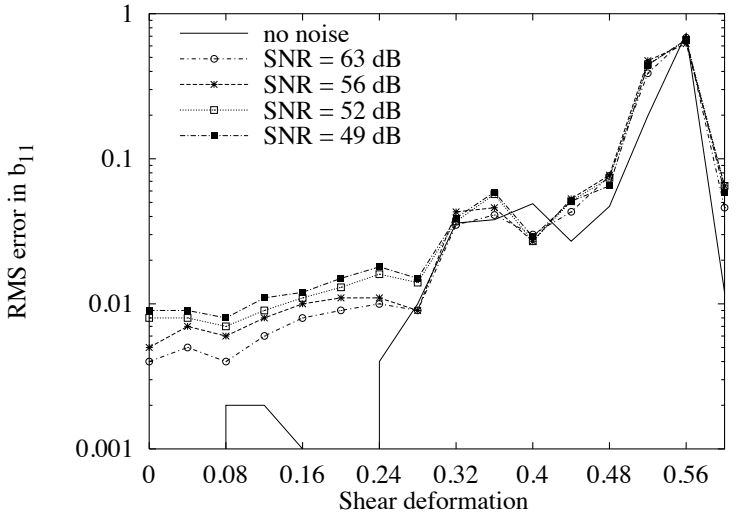



**Figure 6.35.** RMS error in $b_{11}$ versus shear deformation for the GauArea algorithm as a function of noise.

levels, the RMS error increases substantially. The range is slightly smaller for a SNR of 63 dB but falls to -35 deg. to 30 deg when the SNR $\leq$ 56 dB.

For shears (Figure 6.35) even with no noise the error abruptly rises when the shear deformation is greater than 0.24. For shear deformation less than 0.24, the RMS error is much greater with noise but for values above 0.28 the performance with and without noise

is comparable. The range is the same with and without noise (for all levels of noise $\leq 49$ dB).

Plots for all the affine parameters are shown in Figures C.23(a)-C.25(f) in Appendix C.

### 6.8.2 DGauArea: Noise Performance

The effect of noise on DGauArea is displayed in Figures 6.36-6.38.

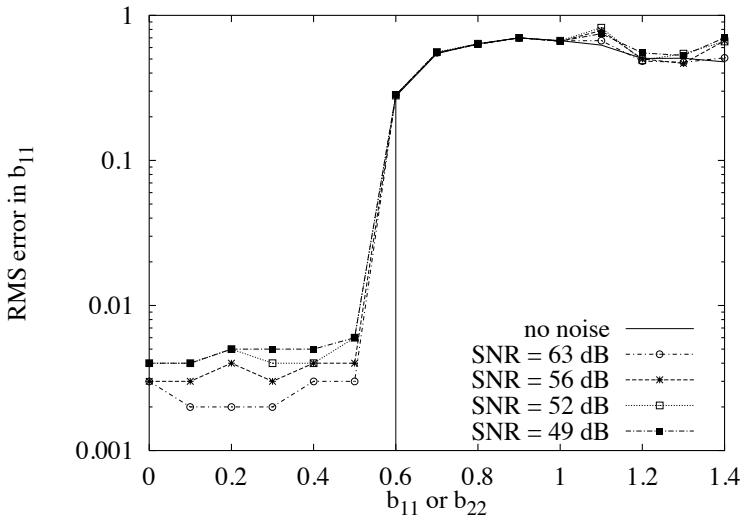



**Figure 6.36.** RMS error in $b_{11}$ versus scale change ($b_{11}$) for the DGauArea algorithm as a function of noise.

For scalings, Figure 6.36 shows that even without noise DGauArea's range is less than 0 to 0.6 while with noise the range is 0 to 0.5. Even with noise added the RMS error for DGauArea is less than 0.06.

For rotations (Figure 6.37), DGauArea has a much smaller range than GauArea even without any added noise. Thus, without noise the range of DGauArea is GauArea's range is from -40 deg. to 45 deg and the RMS error is again less than 0.001 within the effective range. For higher noise levels, the RMS error increases substantially. The range is slightly smaller for a SNR of 63 dB but falls to -35 deg. to 30 deg when the SNR $\leq 56$ dB.

For shears (Figures 6.32) even with no noise the error abruptly rises when the shear deformation is greater than 0.32. For shear values below 0.32, the performance degrades

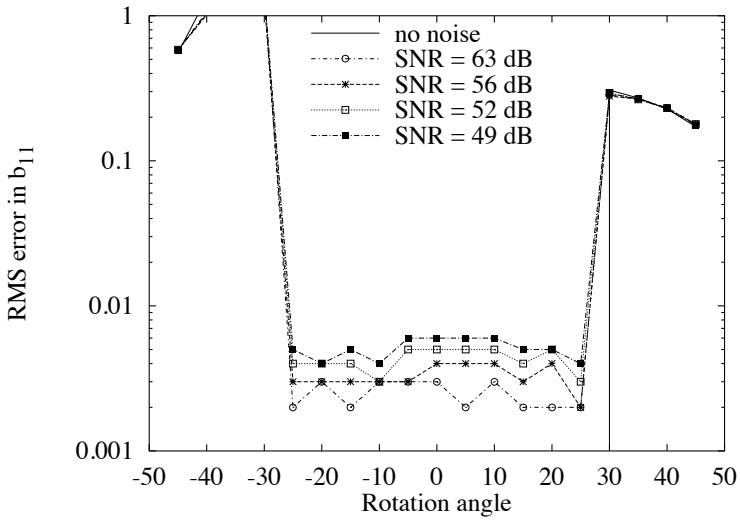



**Figure 6.37.** RMS error in $b_{12}$ versus rotation angle for the DGauArea algorithm as a function of noise.

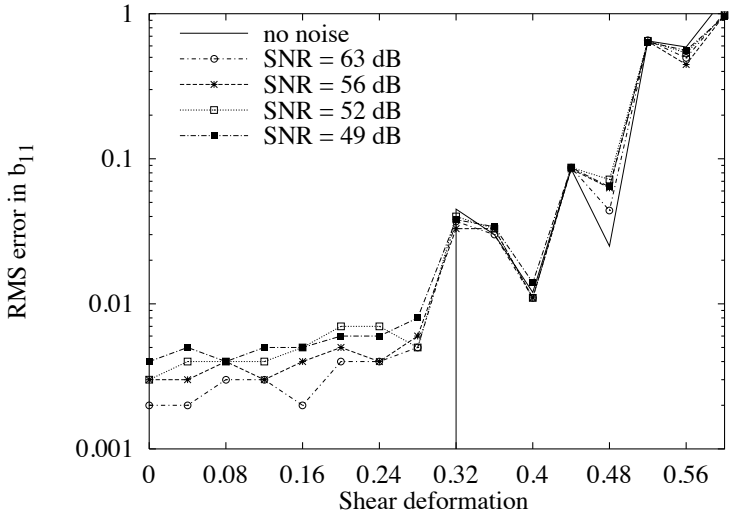



**Figure 6.38.** RMS error in $b_{11}$ versus shear deformation for the DGauArea algorithm as a function of noise.

with even small amounts of noise. Above deformations of 0.32 the performance with and without noise is comparable. The range is the same with and without noise (for all levels of noise $\leq$ 49 dB).

As before, plots for all the affine parameters are shown in Figures C.23(a)-C.25(f) in Appendix C.

## 6.9 Comparison of the Four Algorithms

The four algorithms GauArea, DGauArea, GauAreaN and DGauAreaN (see Figure 6.39) are compared in this section. As in previous sections, the algorithms are compared for scale changes, rotations and shears. (see Figures 6.1-6.3 for examples of these transformations). In addition an experiment was run which compared the performance of the algorithms with the image simultaneously undergoing large multiple transformations. This was achieved by (synthetically) imaging a random dot plane whose slant, tilt and distance from the camera were simultaneously co-varied (see 6.2.1.4 for a detailed explanation and some examples of such transformations).

The experiments show that GauArea has a larger range over which it operates while DGauArea is much more accurate. GauArea has a larger range than GauAreaN while DGauArea has a larger range than DGauAreaN (the additional higher order terms in GauArea and DGauArea are responsible for the larger ranges). In addition, both GauArea and DGauArea converge faster than the corresponding algorithms GauAreaN and DGauAreaN.

In the following experiments, unless otherwise noted, a window of size 13 by 13 was used and every point within this window was used. The filter widths were set to $8\sigma$.

### 6.9.1 Comparison on Scalings

First, the algorithms were compared on scaled images. The first experiment (Figure 6.39; for plots of the other affine parameters see Figures C.26(a)-C.26(f) in Appendix C) was run with two filters (scales $\sigma = 1.25$ and $\sigma = 1.768$).

Clearly, GauArea and GauAreaN have more than double the range of the derivative algorithms DGauArea and DGauAreaN. However, the derivative algorithms are twice as accurate as GauArea and GauAreaN. GauArea has the largest range $b_{11} = (0, 1.2)$, i.e. $a_{11} = (1.0, 2.2)$. GauAreaN has a range of $b_{11} = (0, 1.0)$. DGauArea's range is $b_{11} = (0, 0.5)$ while DGauAreaN has a range of $b_{11} = (0, 0.4)$.

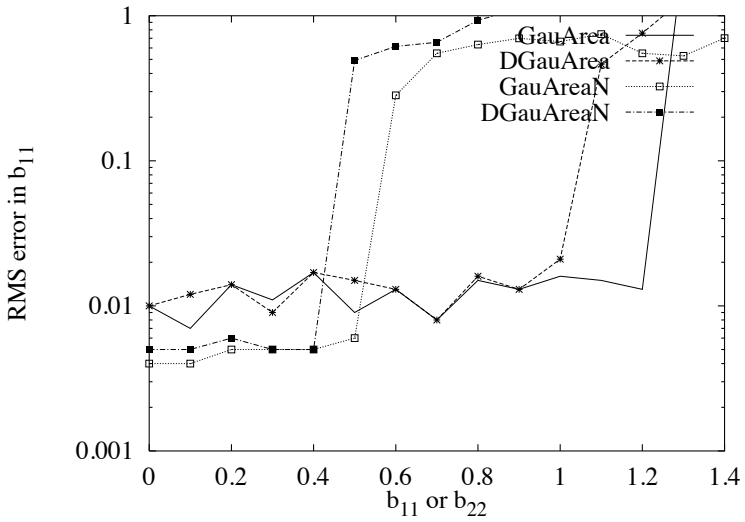



**Figure 6.39.** RMS error in $b_{11}$ versus scale change ($b_{11}$) for four different algorithms for filter scale 1.25.

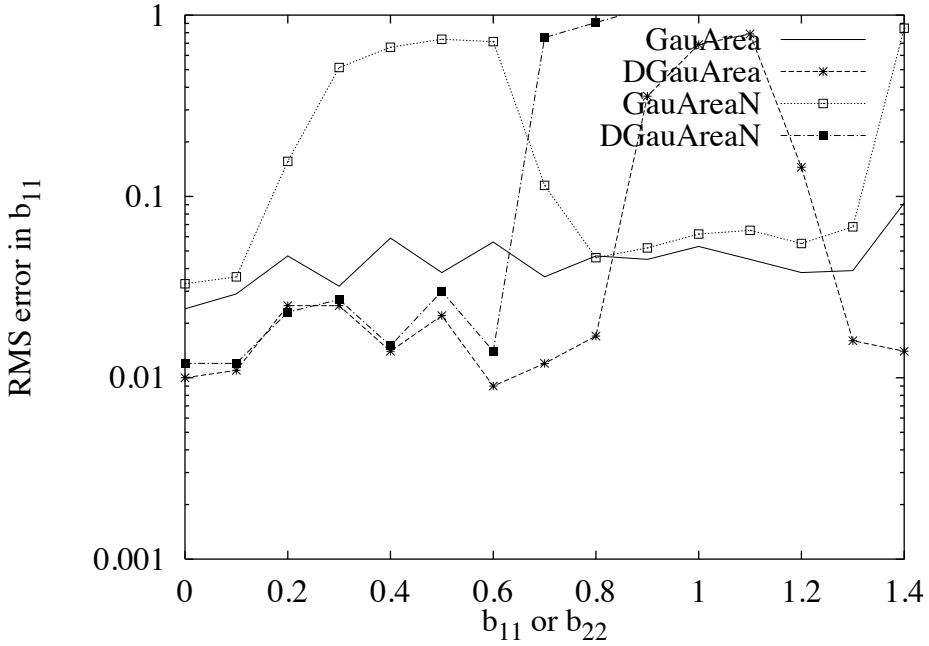



**Figure 6.40.** RMS error in $b_{11}$ versus scale change ($b_{11}$) for four different algorithms for filter scale 1.768.

Since GauAreaN does not model the deformation, its performance deteriotates with increasing filter scale (see Figure 6.40 which uses filters at $\sigma = 1.768$ and $\sigma = 2.5$). In Figure 6.40 the error for GauAreaN rises and then falls, but we assume (see discussion in

Section 6.2.2) that for values of $b_{11} \geq 0.1$ it is unstable. Thus, the range of GauAreaN may be assumed to be $b_{11} < 0.1$. The poor performance of GauAreaN at larger scales is to be expected, since it is more critical to model the deformation of the Gaussian when its standard deviation is larger. The derivative algorithm DGauAreaN, which also does not model the deformation does not suffer any significant degradation with scale. Other experiments (not shown here) show that at much higher scales, DGauAreaN suffers more degradation than DGauArea (the latter models deformations).

For all other algorithms, although their accuracy goes down with scale (probably due to poorer localization), their range increases.

### 6.9.2 Comparison on Rotations

This experiment compares the four algorithms GauArea, DGauArea, GauAreaN and DGauAreaN under rotation (see Figure 6.41; plots for the other affine parameters are shown in Figures C.27(a)-C.27(f) in Appendix C). Two filters, $\sigma = 1.25, 1.768$ were used. Both GauArea and GauAreaN have larger ranges than DGauArea and DGauAreaN. GauArea has the largest range (-35,30) degrees. GauAreaN's range is somewhat smaller (-35,20) degrees. DGauArea has a range of (-25,25) degrees while DGauAreaN has the smallest range at (-25,20) degrees. DGauArea and DGauAreaN are approximately twice as accurate as GauArea and GauAreaN. The performance of the derivative algorithms DGauArea (see 6.5 for experiments on DGauArea) and DGauAreaN (experiments not shown) improves with scale (see 6.5 for experiments on DGauArea) while the performance of GauAreaN drops dramatically with scale.

### 6.9.3 Comparison on Shear Deformations

This experiment compares the four algorithms GauArea, DGauArea, GauAreaN and DGauAreaN on shear deformations (see Figure 6.42 and Figures C.28(a)-C.28(f) in Appendix C. Two filters, $\sigma = 1.25, 1.768$ were used. The range for all four algorithms is

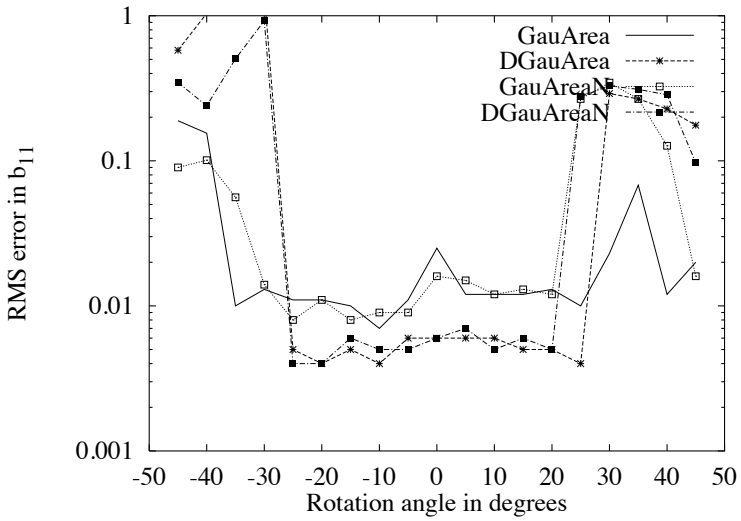



**Figure 6.41.** RMS error in $b_{11}$ versus rotation angle for four different algorithms.

roughly the same. However, the derivative algorithms are twice as accurate as GauArea and GauAreaN.

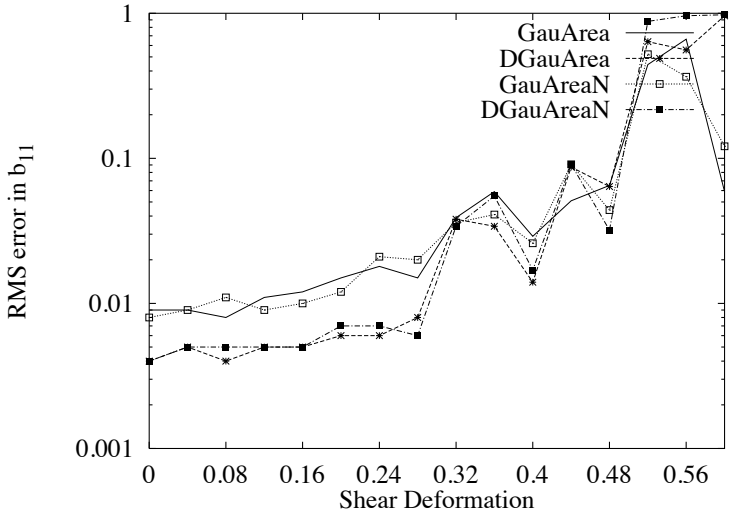



**Figure 6.42.** RMS error in $b_{11}$ versus shear deformation for four different algorithms.

### 6.9.4 Comparison on a Plane Moving in Space - Multiple Covarying Parameters

The next experiment (Figure 6.43) is conducted by assuming that a plane covered with a random dot pattern is moving in space and its slant, tilt and distance from the camera

co-vary simultaneously. The details of how the images were created is discussed in Section 6.2.1.4. It suffices to note here that the slant $\alpha$ was varied from 0 deg. to 56.6 deg ($\cos(\alpha)$ was varied from 1.0 to 0.55 in steps of 0.05). The scale s was simultaneously co-varied from 1.0 to 1.9 in steps of 0.1 and the angle $\beta$ was co-varied from 0 deg. to 45 deg. in steps of 5 deg.

If two filters ($\sigma = 1.25, 1.768$) are used, the Gaussian algorithms GauArea and GauAreaN converge for all transformations while the derivative algorithms converge when the slant is less than 49 degrees ($cos(\sigma) \geq 0.65$). However, the derivative algorithms have less than half the error of the Gaussian algorithms.

For comparison, plots of all the affine parameters are shown in Figures C.29(a)-C.29(f) in Appendix C.

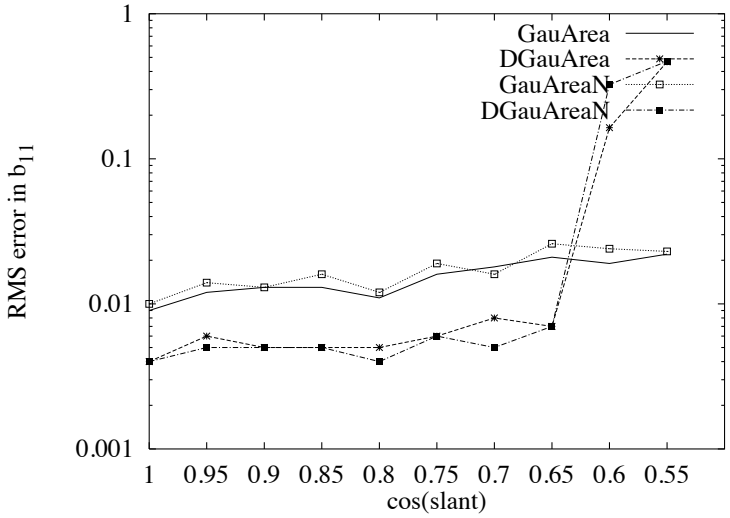



**Figure 6.43.** Comparison RMS errors for four different algorithms for a plane moving in space. The error in the affine parameters is plotted against the cosine of the slant angle. The scale s was simultaneously co-varied from 1.0 to 1.9 in steps of 0.1 and the angle $\beta$ was co-varied from 0 deg. to 45 deg. in steps of 5 deg.

If the filter scales are increased to ($\sigma = 2.5, 3.54$) then the performance of GauAreaN drops dramatically while the other three algorithms work over the entire range of transformations. The derivative algorithms still perform better than the Gaussian algorithm in terms of error (although the error for all algorithms is higher than in the previous experiment).

## 6.10 Comments on Experiments with Synthetic Images

The results from the experiments are briefly summarized here. The Gaussian and its derivatives can be discretized by sampling the continuous functions, provided the filter's scale is not too small ($\geq 0.5$ pixels). The discretized filters may be truncated but the truncation radius must be at least $\pm 4\sigma$.

For small values of scale, GauArea has the largest range followed by GauAreaN. The derivative algorithms DGauArea and DGauAreaN have much smaller ranges especially for scalings and rotations. As the scale is increased, the range of the derivative algorithm improves dramatically and they perform better than GauArea and GauAreaN. The derivative algorithms are more accurate over their effective range by at least a factor of 2 over algorithms which use only the Gaussian equation.

It is also seen that as filter scales are increased, deformation is more important (see Section 6.11 for a more detailed discussion). Thus, GauAreaN fails for filter scales which are 1.768 or larger while GauArea works even for much larger filter scales. The derivative algorithm DGauAreaN breaks down only at really large filter scales ($\approx 5$). DGauArea on the other hand works quite well even at such large scales.

The experiments also show that the algorithms are able to recover the affine transform when multiple simultaneous transformations are involved. GauAreaN as before does poorly when the filter scales are large, but the other three algorithms perform quite well over a range of filter scales.

## 6.11 Tests with Real Images

The performance of the algorithms was also tested with real images in a number of experiments. Each experiment was performed on a pair of real images; the second image in the pair was obtained by either moving the camera or by moving one of the objects in the scene and keeping the camera stationary.

The experiments were carried out on scenes containing a wide variety of objects and different kinds of surfaces. The six experiments conducted were:

1. Face Images (Section 6.11.1): A human face is clearly not planar. But for small motions, frontal views of the face can be well approximated by an affine transform.

2. Monet Images (Section 6.11.2): The scene is a poster by the artist Monet. The scene is planar and is richly textured. In this sense, it has information almost everywhere (in the textured areas) like the synthetic random dot images used before. The transformation between the two images has a large scale change as well as shear.

3. Bookcase Images (Section 6.11.4): The books in the bookcase may be approximated by a planar surface. However, there is a strong perspective effect - notice that lines parallel to adjacent shelves appear to converge in in the picture. There is considerable rotation between the two images.

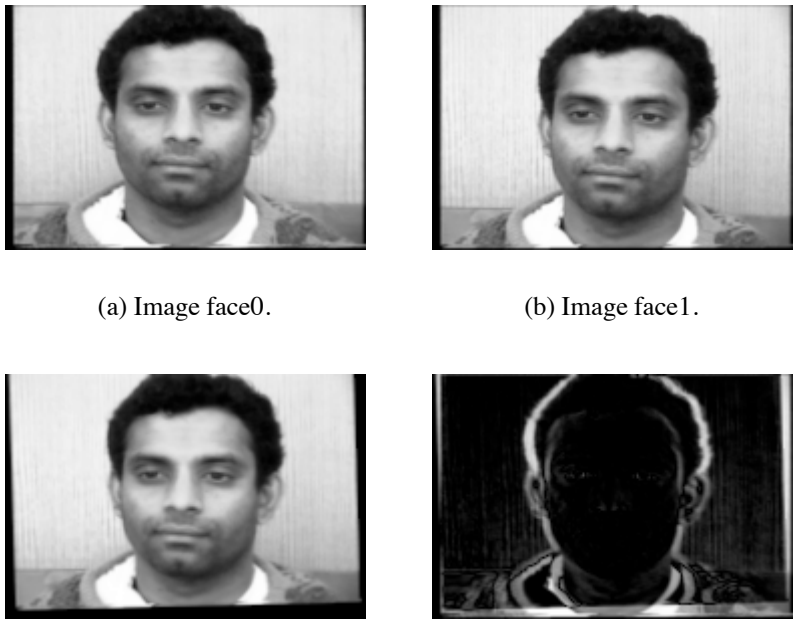
4. Blackboard Images (Section 6.11.3): The scene includes a blackboard and some books in the foreground. The blackboard forms one planar object while the books in the foreground clearly cannot be modelled using the same planar object. The two images differ by a large scale change and a rotation.

5. Pepsi Can Images (Section 6.11.5): A pepsi can is cylindrical. Locally, however, it may be well approximated as an affine. Three images of the pepsi can are shown. Between the first two, there is a large scale change. Between the first and third pepsi images, there is both a large scale change, a rotation as well as some shear deformation.

6. Elephant Images: These two frames of an elephant were obtained from a video sequence of a moving elephant. The elephant's head and body make a large sideways motion. The side of the head forms an approximately planar object and its motion can be modelled as an affine.

### 6.11.1 Face Images

In this section, we demonstrate that images of objects which are not strictly planar can also be matched using the algorithms. A picture of a face was taken from a roughly frontal position Figure 6.44(a). A second picture was taken from a slightly different viewpoint Figure 6.44(b). Clearly, the face is a non-planar object.



(a) Image face0.

(b) Image face1.

(c) Face1 warped using recovered affine transform.

(d) Difference between a and c.

**Figure 6.44.** Matching of face0 and face1 using algorithm DGauArea.

No prior translation estimates were provided in this case. Instead the algorithms were run at the center of the image. All the algorithms gave roughly the same result. Figure 6.44(c) is obtained by warping face1 according to the resulting affine transform. Figure 6.44(d) is the difference between face0 and the warped version of face1. Notice the excellent registration between the two faces. The white areas are due to portions on the body or the mid-section of the head which cannot be modelled using the same plane as the face.

Table 6.2 shows the affine parameters computed by DGauArea Similar results were obtained using the other algorithms. The good quality of the registration results obtained

**Table 6.2.** Recovered affine parameters for Face Image.

| Algorithm | $a_{11}$ | $a_{12}$ | $a_{21}$ | $a_{22}$ | $t_x$ | $t_y$ |
|---|---|---|---|---|---|---|
| DGauArea | 0.980 | -0.016 | 0.020 | 0.990 | -2.02 | 4.64 |

using all the algorithms is not surprising because of the small affine transformation (see Table 6.2) between the two images.

The registered image in Figure 6.44(c) was obtained with window sizes of 41 by 41. Similar results were obtained with smaller window sizes. With larger window sizes, the registration quality was poorer. The importance of not using large patches is emphasized by this result.

### 6.11.2 Monet Images

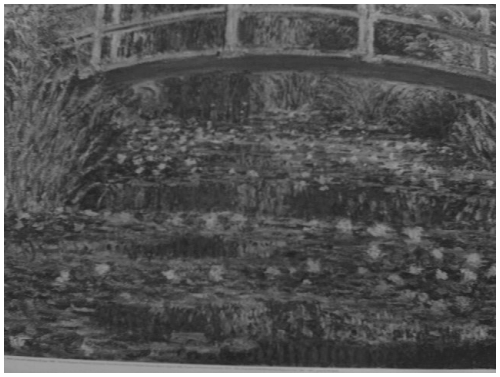
This experiment is performed on a pair of images of a flat Monet poster. One image (Monet1 Figure 6.45(b)) was taken frontally while the other image (Monet0 Figure 6.45(a)) was taken from the side. There are large scale change as well as shear between the two images.

The algorithms were applied at the point shown in Figure 6.45(a). Figure 6.45(c) shows the image obtained by warping Monet1 using the recovered affine transform (see Table 6.3 - this was obtained with a window size of 41 by 41). Figure 6.45(d) shows the difference between the warped image and the first image Monet0. Notice that the region used for running the algorithms is well registered. Other areas of the image are not perfectly registered (although for example the bridge is in the right place). The imperfect registration is due to the steep slant of the first image with respect to the second.

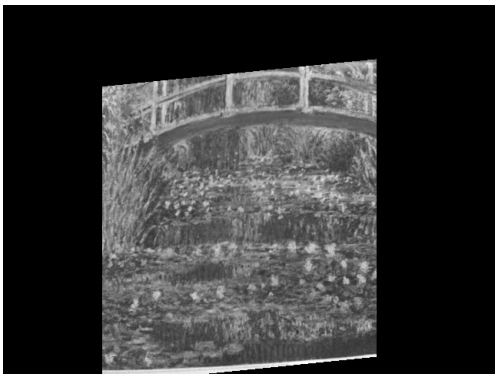
Table 6.3 lists the affine parameters obtained for the algorithm DGauArea. From the table, we can estimate that the two images differ by a shear angle of roughly 6 deg, a shear deformation of 0.256 and a scale change of 0.67.

Table 6.4 shows how the four different algorithms compare. The table lists, for a particular window size, the range of filter scales for which each algorithm produces good results.

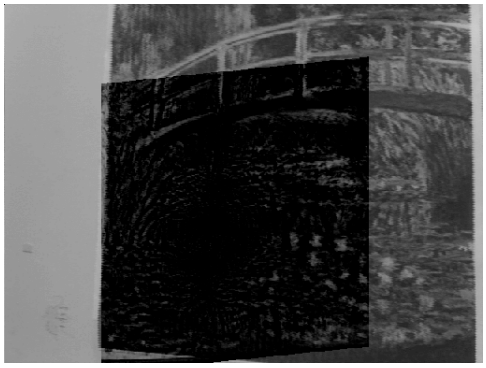
(a) Image Monet0 with point marked on it.



(b) Image Monet1.



(c) Monet1 warped using recovered affine transform.



(d) Difference between a and c.

**Figure 6.45.** Matching of Monet0 and Monet1 using algorithm DGauArea.

For example, at window size 41 by 41 DGauArea converges for a range of filter scales from $\sigma = 2$ to $\sigma = 5$. Only filter sizes from $\sigma = 2$ to $\sigma = 5$ were tried. From the table it is clear that DGauArea is the most robust algorithm in the sense of producing good results over the widest range of scales. The other algorithms in order of robustness are DGauAreaN, GauArea and GauAreaN.

### 6.11.3 Blackboard Images

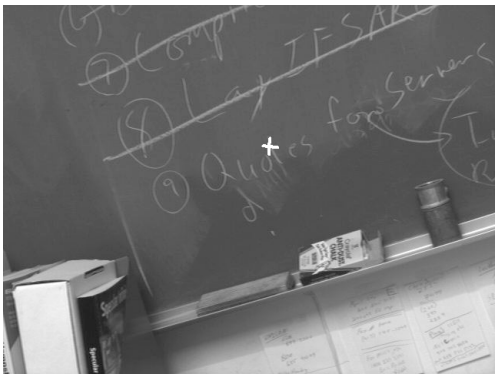
The algorithms work well with planar surfaces. This is demonstrated by the following experiment conducted on an office scene containing a planar blackboard along with some

**Table 6.3.** Recovered affine parameters for the Monet images.

| Algorithm | $a_{11}$ | $a_{12}$ | $a_{21}$ | $a_{22}$ | $t_x$ | $t_y$ |
|---|---|---|---|---|---|---|
| DGauArea | 0.559 | 0.000 | 0.056 | 0.809 | 9.57 | -37.56 |

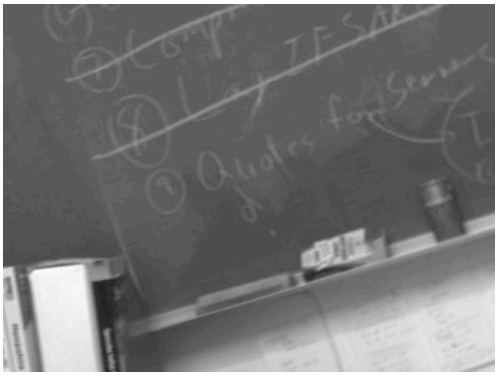
**Table 6.4.** Effective performance in terms of filter size for four different algorithms on the Monet images.

| Window Size | GauArea | GauAreaN | DGauArea | DGauAreaN |
|---|---|---|---|---|
| 41 x 41 | Poor results | Poor results | 2 to 5 | 2 to 5 |
| 81 x 81 | 3.54 to 5 | 2 to 5 | 3.54 to 5 | 5 |

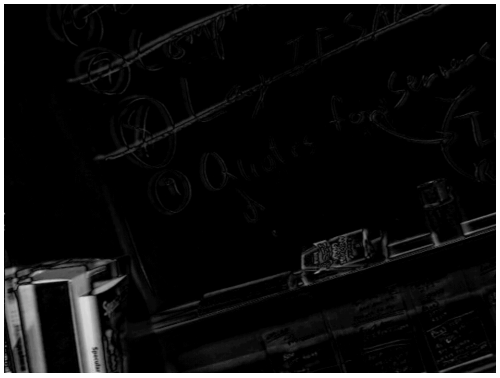


(a) Image blackboard0 with point marked on it.

(b) Image blackboard1.



(c) Blackboard1 warped using recovered affine transform.



(d) Difference between a and c.

**Figure 6.46.** Matching of blackboard0 and blackboard1 using algorithm DGauArea

**Table 6.5.** Effective performance in terms of filter sizes for four different algorithms on the Board images.

| Window Size | GauArea | GauAreaN | DGauArea | DGauAreaN |
|---|---|---|---|---|
| 11 x 11 | Does not converge | Does not converge | Does not converge | Does not converge |
| 21 x 21 | 2 | Does not converge | 3 to 5 | Does not converge |
| 41 x 41 | 2 to 5 | 2 to 3 | 3 to 5 | 5 |
| 61 x 61 | 3 to 5 | Does not converge | 5 | 5 |
| 81 by 81 | 5 | Does not converge | Does not converge | Does not converge |

non-planar objects. Figure 6.46(a) (referred to as blackboard0) is the original image with a point marked on it. Figure 6.46(b) (referred to as blackboard1) is a second image taken from a different viewpoint. There is a large scale change (roughly a factor of 2) as well as a rotation of 18 degrees.

A translation estimate is provided (see Appendix B for an example of how to obtain an estimate). Figure 6.46(c) shows the image obtained by affine warping blackboard1 by the recovered affine transform (see Table 6.5); the window size used was 41 by 41. Figure 6.47(d) is the difference between the warped image of blackboard1 and blackboard0. The blackboard is well registered. The surfaces of objects right next to the blackboard are registered well if these surfaces are roughly parallel to the blackboard's surface. For example the duster and the tin can are registered well. On the other hand, the top surface of the box of chalk shows some mis-registration. The books are sufficiently far away from the blackboard that the same affine transformation does not model their deformation, hence they are not registered.

Table 6.5 lists, for a given window size, the range of filter scales for which the different algorithms give good results. Filter scales from $\sigma = 2$ to $\sigma = 5$ were tried. It is clear that GauAreaN performed poorly - it worked only when the window size was 41 x 41. The best performance is shown by the algorithms which account for deformation i.e. GauArea

and DGauArea. It is to be noted that GauArea tends to work for smaller filter scales while DGauArea works better when the filter scales are larger.

### 6.11.4 BookCase Images

(a) Image bookcase0 with point marked on it.

(b) Image bookcase1.

(c) Bookcase1 warped using recovered affine transform.

(d) Difference between a and c.

**Figure 6.47.** Matching of bookcase0 and bookcase1 using algorithm DGauArea.

The algorithms work well even in situations where there are strong perspective effects and reasonably large rotations. The experiment was performed on images of a bookcase which shows strong perspective effects. Figure 6.47(a) (referred to as bookcase0) shows the original image with a point marked on it. Figure 6.47(b) (referred to as bookcase1) is a second image taken from a different viewpoint. There is about 21 degrees of rotation

**Table 6.6.** Recovered affine parameters for the bookcase images.

| Algorithm | $a_{11}$ | $a_{12}$ | $a_{21}$ | $a_{22}$ | $t_x$ | $t_y$ |
|---|---|---|---|---|---|---|
| DGauArea | 0.958 | 0.372 | -0.368 | 0.924 | -28.82 | 29.46 |

**Table 6.7.** Recovered affine parameters for Pepsi0 and Pepsi1 around point 1.

| Algorithm | $a_{11}$ | $a_{12}$ | $a_{21}$ | $a_{22}$ | $t_x$ | $t_y$ |
|---|---|---|---|---|---|---|
| GauArea | 1.41 | -0.01 | -0.067 | 1.36 | 25.08 | 9.24 |
| DGauArea | 1.40 | -0.02 | -0.067 | 1.38 | 25.08 | 9.30 |

between the two images as well as small amounts of other deformations. Notice that some of the parallel lines in the image converge, indicating a strong perspective effect.

For a window size of 81 by 81, all the algorithms converged for scales from $\sigma = 1.25$ to $\sigma = 3.54$. However, for scales of $\sigma = 5$, only GauArea and DGauArea converged on this image but not GauAreaN or DGauAreaN i.e. for large scales the algorithms using deformation converged. Figure 6.47(c) shows the image obtained by affine warping bookcase1 by the recovered affine transform (see Table 6.6 obtained using $\sigma = 5$). Figure 6.47(d) is the difference between the warped image of bookcase1 and bookcase0. The registration is good but the strong perspective effect is noticeable in the white areas on some of the books.

### 6.11.5 Pepsi

In this section, tests on real images of a Pepsi can are shown. The Pepsi can images are the same ones that were used to test the gross translation algorithm in Appendix B.

The first experiment uses images Pepsi0 (Figure 6.48(a)) and Pepsi1 (Figure 6.48(b)). Pepsi1 was created by moving the camera towards the Pepsi can. The region around the part marked in Figures 6.48(a) is sought to be matched. The can is cylindrical and, therefore, the transformation between two images of the entire Pepsi can cannot be described by a single affine transform. Note also that the transformation between the two images of the background is considerably different from that of the Pepsi can. However, a part of the can may be described by using a single affine transform.

(a) Image Pepsi0 with Point 0 marked on it.

(b) Image Pepsi3.

(c) Pepsi0 warped using recovered affine transform.

(d) Difference of b and c

**Figure 6.48.** Matching of Pepsi0 and Pepsi1 using algorithm GauArea

A point (Point 0) is marked on the Pepsi0 image. The algorithm in Section B is used to recover an estimate of the translation. Using this estimate, all four algorithms (GauArea, GauAreaN, DGauArea and DGauAreaN) were run on the images. As the first row of Table 6.8 shows, for this pair of images, all algorithms converged for a wide set of parameters.

The following pairs of scales ($\sigma = 1.25$ and $\sigma = 1.25\sqrt{2} = 1.768$) were used. The first row of Table 6.8 lists the window sizes for which the different algorithms converged. GauArea converged for window sizes ranging from 11x11 [3] to more than 173x173 while GauAreaN converged for window sizes from 13x13 to more than 173x173. Thus GauArea performed slightly better than GauAreaN. The derivative algorithms needed larger window sizes to converge. DGauArea and DGauAreaN converged for window sizes ranging from 21x21 to larger than 173x173. However, the images were not properly registered by DGauAreaN unless the windows were at least of size 31x31. Thus, DGauArea which accounts for deformations performs better than DGauAreaN.

Note that the Pepsi image regions were correctly registered only at the smaller window sizes. At many of the large window sizes, large portions of the image which do not belong to the Pepsi can are included. These portions are further from the camera and the transformations they undergo due to viewpoint change are considerably different. Thus, at large window sizes, the Pepsi can image regions were not properly registered.

Figure 6.48(c) shows the image Pepsi0 warped by the recovered affine transform while Figure 6.48(d) shows the difference between this warped image and the Pepsi1 image. Notice that the region around Point 0 is well described by the affine transform while portions of the can much further away are not described as well. These images were created using GauArea with a window size of 23x23.

The affine parameters recovered by the algorithm GauArea are listed in Table 6.7. Table 6.7 also lists the affine parameters recovered using DGauArea with the same window sizes.

The second experiment is done using images Pepsi0 (Figure 6.49(a)) and Pepsi3 (Figure 6.49(b)). Notice the considerable rotation of Pepsi3 with respect to Pepsi0. There is also a shear component caused by an artifact of the imaging process. The camera's scale in the

[3] All window sizes in pixel units.

**Table 6.8.** Effective performance in terms of filter sizes for four different algorithms on the Pepsi images.

| Transformation | GauArea | GauAreaN | DGauArea | DGauAreaN | Scales Used |
|---|---|---|---|---|---|
| Scale Change $\approx 1.4$ | 11 x 11 to v. large | 13 x 13 to v. large | 21 x 21 to v. large | 31 x 31 to v. large | 1.25 and 1.77 |
| Scale Change + 45 def rotation + shear | 25 x 25 to 31 x 31 | Does not converge | 21 x 21 to 25 x 25 | 21 x 21 to 25 x 25 | 2.5 and 3.54 |
| | Converges | Does not converge | Converges | Does not Converge | 3.54 and 5.0 |
| | 21 x 21 | Does not converge | 19 x 19 to 39 x 39 | Does not Converge | 5.0 and 7.07 |

**Table 6.9.** Recovered affine parameters for Pepsi0 and Pepsi3 around Point 3.

| Algorithm | $a_{11}$ | $a_{12}$ | $a_{21}$ | $a_{22}$ | $t_x$ | $t_y$ |
|---|---|---|---|---|---|---|
| GauArea | 1.26 | -0.73 | 1.06 | 1.13 | 0.90 | 30.30 |
| DGauArea | 1.27 | -0.69 | 1.10 | 1.15 | 0.98 | 30.27 |

x and y direction are in the ratio of 0.7/1. Thus, rotating the camera causes shear between two images.

The region around the part (point 3) marked in Figures 6.49(a) is sought to be matched. The algorithm in Appendix B is again used to recover an estimate of the translation. Using this estimate, all four algorithms (GauArea, GauAreaN, DGauArea and DGauAreaN) were run on the images. As Table 6.8 shows GauAreaN did not converge for any set of parameters that were used. As the table clearly shows, DGauArea works over the widest range of window sizes and filter scales followed by GauArea. DGauAreaN works only for one set of filter scales. [4]

Figure 6.49(c) shows the image Pepsi0 warped by the recovered affine transform while Figure 6.49(d) shows the difference between this warped image and the Pepsi3 image. Notice that the region around Point 3 is well described by the affine transform while portions

[4]The recovered affine transform tends to be less accurate at the larger window sizes because for large window sizes, the contribution of portions of the image outside the can is significant.

(a) Image Pepsi0 with Point 3 marked on it.

(b) Image Pepsi3.

(c) Pepsi0 warped using recovered affine transform.

(d) Difference of b and c.

**Figure 6.49.** Matching of Pepsi0 and Pepsi3 using algorithm GauArea

of the can much further away are not described as well. These images were produced using algorithm GauArea with a window size of 25x25.

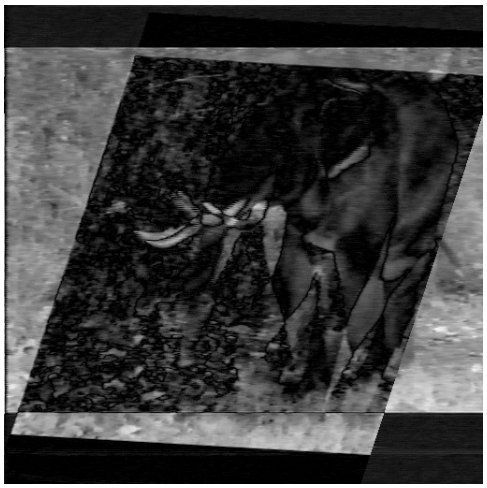
The affine parameters recovered by the algorithm GauArea are listed in Table 6.9. Table 6.9 also lists the affine parameters recovered using DGauArea with the same window sizes.

### 6.11.6 Elephant

(a) Image elephant0 with point marked on it.

(b) Image elephant1.

(c) Elephant1 warped using recovered affine transform.



(d) Difference between a and c.

**Figure 6.50.** Matching of elephant0 and elephant1 using algorithm DGauArea

An experiment was performed on images of a elephant obtained from a video (Figure 6.50(a),referred to as elephant0) is the the original image with a point marked on it. Figure 6.50(b) (referred to as elephant1) is the second image obtained after the elephant has moved sideways. Although the elephant is not planar, the side of the head can be modelled as a

**Table 6.10.** Recovered affine parameters for the elephant images.

| Algorithm | $a_{11}$ | $a_{12}$ | $a_{21}$ | $a_{22}$ | $t_x$ | $t_y$ |
|---|---|---|---|---|---|---|
| DGauArea | 0.755 | 0.343 | -0.043 | 1.036 | 44.23 | -23.91 |

planar surface and matched using an affine transform. Note that a sideways motion amounts to a shear deformation.

A translation estimate was provided (see Appendix B) on how to generate such estimates). On this pair of images, only DGauArea successfully registered images. Figure 6.50(c) shows the image obtained by affine warping elephant1 by the recovered affine transform (see Table 6.10). Figure 6.50(d) is the difference between the warped image of elephant1 and elephant0. The elephant's head is well registered except for the tusks, as expected. However, the rest of the body is not since they are clearly not part of the same planar approximation.

Table 6.10 shows the affine parameters recovered by the DGauArea algorithm using a window size of 41 x 41. From these deformation parameters it can be estimated that there is a shear deformation between the two images specified by a deformation angle of 10.5 deg. and a deformation of 0.16.

### 6.11.7 Comments on the Real Image Experiments

The experiments show that when large affine transforms must be recovered, accounting for deformations is important. The experiments also show that deformations play a much more crucial role when the affine transforms are composed of shears or rotations. The importance of using small window sizes is emphasized by the fact that the algorithms may not converge when large window sizes are involved.

The experiments shows that as the filter scale is increased, it is essential to account for deformation; the break even point depends on the order of the derivative. Thus, for algorithms which linearize the Gaussian (GauArea and GauAreaN), deformation must be accounted for at filter scales as small as $\sigma = 1.768$. Algorithms which linearize the first

derivative of the Gaussians must take deformation into consideration when the filter scales exceed $\sigma = 3.54$.

Most of the energy of the Gaussian filter is concentrated within $\pm 2\sigma$ of the origin. Thus for small affine transforms and small scales, the deformation is barely noticeable. For example, consider a similarity transform with a scale change of 1.2. Also, consider a Gaussian of scale 1.0 pixel. Due to the scale change, the second image needs to be filtered with a Gaussian of scale $1.2 \times 1.0 = 1.2$ pixels. At a distance of about $2\sigma$, the change is less than $1.2 \times 2 - 1 \times 2 = 0.4$ pixels. Since the change is less than 1 pixel, the deformation does not need to be accounted for. As the filter scale or the affine transform is increased, deformation becomes increasingly important.

The derivative algorithms show similar behaviour. Thus, DGauAreaN breaks down at larger filter scales (although these are much larger than the ones for GauAreaN) because it does not account for filter deformations. DGauArea on the other hand is well behaved even for large filter scales.

Thus, one may conclude that accounting for filter deformations gives rise to better algorithms than when the deformations are not taken into consideration. The best performance on real images is shown by DGauArea followed by GauArea - both algorithms which account for deformations. DGauArea tends to work better when larger filter widths are used while GauArea works better for smaller filter widths. The algorithm DGauAreaN comes next in performance, while the algorithm GauAreaN often performs poorly. In the next chapter we show how DGauArea performs on a real application.

# CHAPTER 7

# APPLICATIONS OF AFFINE MATCHING ALGORITHMS

## 7.1 Introduction

Matching images under affine transforms is useful in many practical applications. These include registration, bootstrapping visual tracking, the detection of objects and logos and the indexing of historical manuscripts (word spotting). Methods for recovering large affine transforms may be used to bootstrap the visual tracking process. Most of the time visual tracking involves finding matches between closely spaced frames. However, in certain situations, closely spaced frames may not be available or may have been lost. In such cases, images between which substantial motion of the camera has occurred may need to be matched. The experiments on real images of the Pepsi can described in Chapter 6 show how the Pepsi can may be tracked even over large affine changes. Other applications where widely spaced frames need to be matched include the detection of objects or logos. The experiments on the Pepsi can again illustrate the potential for detecting objects and logos. The GauArea algorithm derived here has also been applied to the problem of matching a pair of images where one of them is defocused [51].

Affine matching has also been used for the problem of indexing historical manuscripts written by the same person [45, 44, 42]. The index is prepared by finding words similar to a given word. The set of similar words is obtained by matching word images. This process has been called "word spotting" [45, 44, 42], since it is similar to techniqes in speech for locating words by matching them [29]. Previously, [42] two algorithms, Euclidean Distance Matching (EDM) which matches words under translation, and an algorithm by Scott and Longuet-Higgins [61] (see Chapter 2 for a description of the latter) which matches words

under an affine transformation, have been applied to the problem and it has been shown that the affine matching algorithm performs well on the standard information retrieval measures of recall and precision (see Section 7.5.3 for definitions of recall and precision).

The rest of this chapter is devoted to the discussion of the application of affine matching to the problem of word spotting. First, a brief motivation of the problem is provided (most of this material is taken from [42]). Then the matching aspects of the paper are briefly discussed and it is shown how the Scott and Longuet Higgins (SLH) [61] algorithm, and the DGauArea algorithm developed in Chapter 5, may be applied to the problem. Note that most of the discussion in this chapter is confined to the matching aspects of this process and the reader is referred to [42] for other aspects. The difficulty of the matching process may be seen by the following statistic. If one page is to be indexed, there are a possible $words-per-page^2$ matches where $words-per-page$ is the number of words in a page. Thus, if there are 300 words in a page, the number of possible matches is roughly 90,000 for just one page (this number may be pruned using certain heuristics). It is shown that DGauArea performs better than SLH in terms of speed, recall and precision and stability.

## 7.2 Motivation

There are many historical manuscripts written in a single hand which it would be useful to index. Examples include the W. B. DuBois collection at the University of Massachusetts, Margaret Sanger's collected works at Smith College and the early Presidential libraries at the Library of Congress. These manuscripts are largely written in a single hand. Such manuscripts are valuable resources for scholars as well as others who wish to consult the original manuscripts and considerable effort has gone into manually producing indices for them. For example, a substantial collection of Margaret Sanger's work has been recently put on microfilm (see http://MEP.cla.sc.edu/Sanger/SangBase.HTM) with an item by item index. These indices were created manually. The indexing scheme proposed here will help in the automatic creation and production of indices and concordances for such archives.

One solution is to use Optical Character Recognition (OCR) to convert scanned paper documents into ASCII and then use a text based retrieval engine [70]. Existing OCR technology works well with standard machine printed fonts against clean backgrounds [8]. It works poorly if the originals are of poor quality or if the text is handwritten. Since Optical Character Recognition (OCR) does not work well on handwriting, an alternative scheme based on matching the images of the words was proposed in [45, 44, 42] for indexing such texts.

Since the document is written by a single person, the assumption is that the variation in the word images will be small. The proposed solution will first segment the page into words and then match the actual word images against each other to create equivalence classes. Each equivalence class will consist of multiple instances of the same word. Each word will have a link to the page it came from. The number of words in each equivalence class will be tabulated. Those classes with the largest numbers of words will probably be stopwords, i.e. conjunctions such as "and" or articles such as "the". Classes containing stopwords are eliminated (since they are not very useful for indexing). A list is made of the remaining classes. This list is ordered according to the number of words contained in each of the classes. The user provides ASCII equivalents for a representative word in each of the top m (say m = 2000) classes. The words in these classes can now be indexed.

The proposed solution completely avoids machine recognition of handwritten words as this is a difficult task [50]. Robustness is achieved compared to OCR systems for two reasons:

1. Matching is based on entire words. This is in contrast to conventional OCR systems which essentially recognize characters rather than words.

2. Recognition is avoided. Instead a human is placed in the loop when ASCII equivalents of the words must be provided.

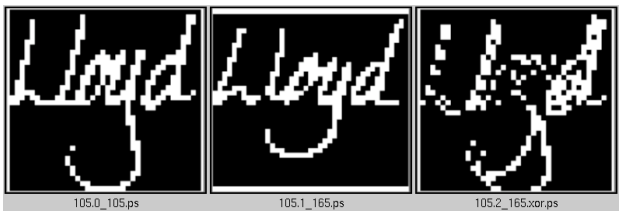
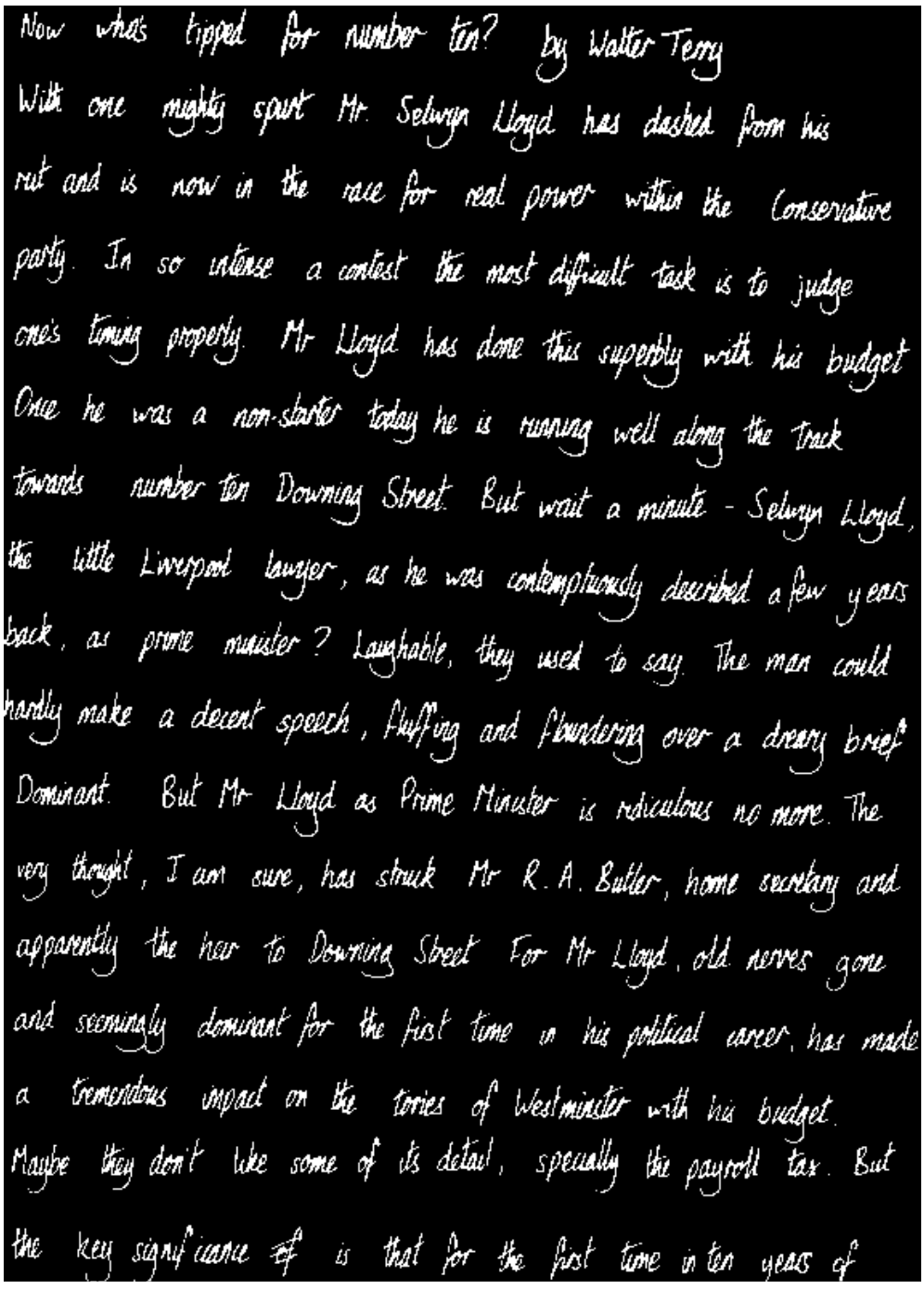



**Figure 7.1.** Two examples of the word “Lloyd” and the XOR image

The matching phase of the problem is the most difficult part of the problem. Image matching of words has been used to recognize words in documents which use machine fonts [32]. Recognition rates are much higher than when the OCR is used directly [32]. Machine fonts are simpler to match than handwritten fonts since the variation is much smaller; multiple instances of a given word printed in the same font are identical except for noise. In handwriting, however, multiple instances of the same word on the same page by the same writer show variations. This variation is difficult to model. Figure (7.1) shows two examples of the word “Lloyd” written by the same person. The last image is produced by XOR’ing these two images. The white areas in the XOR image indicate where the two versions of “Lloyd” differ. This result is not unusual. In fact, the differences are sometimes even larger.

Here, it will be assumed that the transformation between words may be modelled using an affine transform (see [42] which shows that a translational algorithm is insufficient). The actual transformation between words is likely to be more complicated but is difficult to model. However, it is not necessary to recover the correct affine transform for good ranking. Instead, for word spotting, it is important that the words be correctly ranked (in a statistical sense). The accuracy of this ranking is measured by recall and precision.

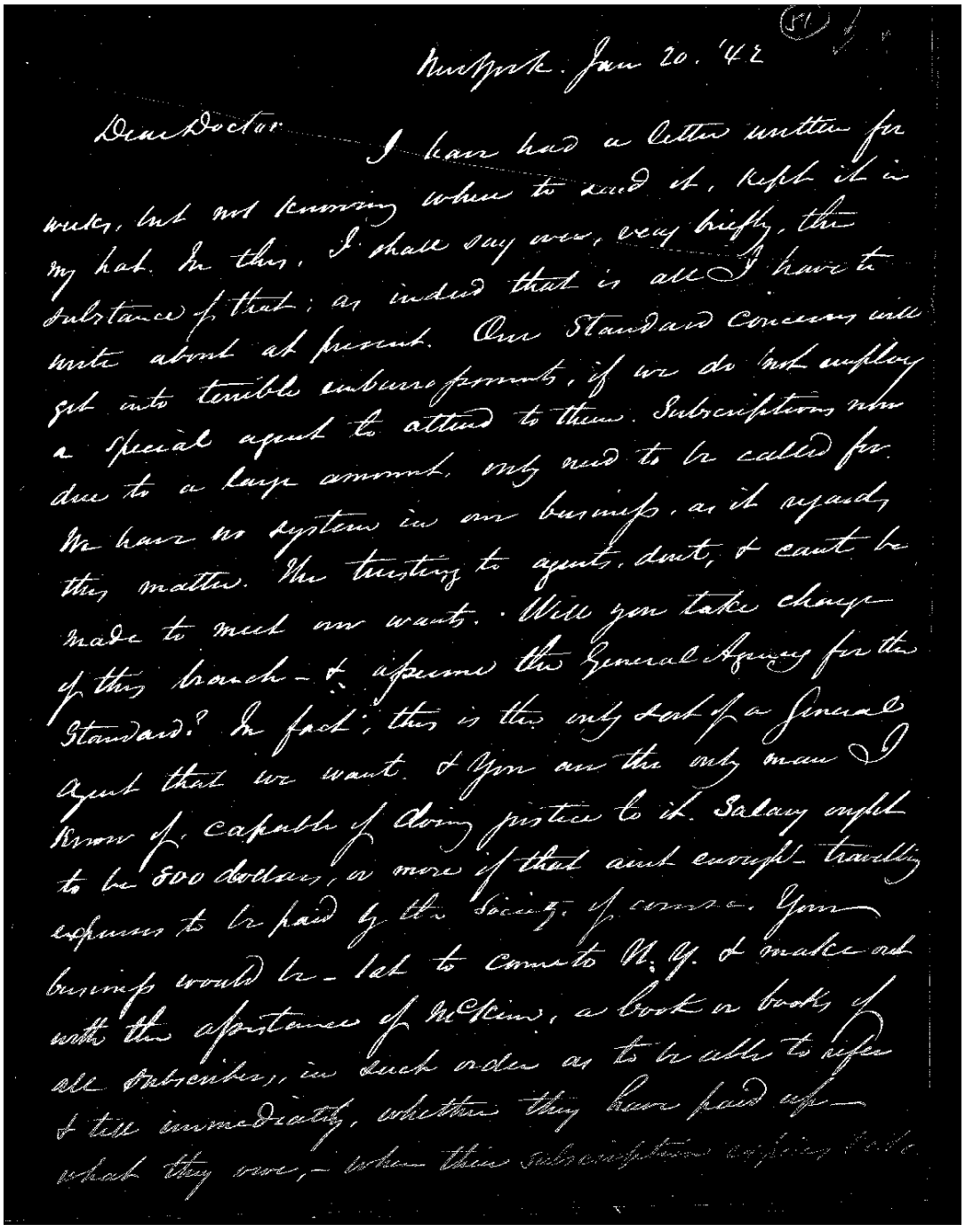
Results on previous experiments with the SLH algorithm [61] are compared with experiments done using DGauArea. Both algorithms model the transformation between words as an affine. The SLH algorithm assumes that two sets of token points are available to it and attempts to find the best affine transformation which relates the two sets of points (see



Now who's tipped for number ten? by Walter Terry

With one mighty spurt Mr. Selwyn Lloyd has dashed from his rut and is now in the race for real power within the Conservative party. In so intense a contest the most difficult task is to judge one's timing properly. Mr Lloyd has done this superbly with his budget. Once he was a non-starter today he is running well along the track towards number ten Downing Street. But wait a minute - Selwyn Lloyd, the little Liverpool lawyer, as he was contemptuously described a few years back, as prime minister? Laughable, they used to say. The man could hardly make a decent speech, fluffing and floundering over a dreary brief. Dominant. But Mr Lloyd as Prime Minister is ridiculous no more. The very thought, I am sure, has struck Mr R. A. Butler, home secretary and apparently the heir to Downing Street. For Mr Lloyd, old nerves gone and seemingly dominant for the first time in his political career, has made a tremendous impact on the tories of Westminster with his budget. Maybe they don't like some of its detail, specially the payroll tax. But the key significance of is that for the first time in ten years of

**Figure 7.2.** The Senior document.



New York. Jan 20. '42

Dear Doctor. I have had a letter written for weeks, but not knowing where to send it, kept it in my hat. In this, I shall say now, very briefly, the substance of that; as indeed that is all I have to write about at present. Our Standard Concern will get into terrible embarrassments, if we do not employ a special agent to attend to them. Subscriptions now due to a large amount, only need to be called for. We have no system in our business, as it regards this matter. The trusting to agents, dont, & cant be made to meet our wants. Will you take charge of this branch - & assume the General Agency for the Standard? In fact; this is the only sort of a General Agent that we want. & You are the only man I know of, capable of doing justice to it. Salary supposed to be 800 dollars, or more if that aint enough. Travelling expenses to be paid by the Society, of course. Your business would be - let to commute N. Y. & make out with the assistance of McClure, a book or books of all Subscribers, in such order as to be able to refer & till immediately, whether they have paid up - what they owe, - when their subscriptions expire &c

Chapter 2). It can handle occlusions as well as cases where points do not have a match. The DGuaArea algorithm treats the word images as greylevel images and then tries to find the best affine match between the two algorithms. DGauArea is not meant to handle occlusions or missing points. Despite that, it will be shown here that DGauArea performs slightly better than SLH on recall precision measures. In addition, an unoptimized version of DGauArea is about 4 times faster than SLH and is more stable.

## 7.3 Outline of Algorithm

1. A scanned greylevel image of the document is obtained.

2. The image is first reduced by half by Gaussian filtering and subsampling (this reduces the processing time).

3. The reduced image is then binarized by thresholding the image.

4. The binary image is now segmented into words. This is done by a process of smoothing and thresholding (see [45, 42]).

5. A given word image (i.e. the image of a word) is used as a template and matched against all the other word images. This is repeated for every word in the document. The matching is done in two phases. First, the number of words to be matched is pruned using the areas and aspect ratios of the word images - the word to be matched cannot have an area or aspect ratio which is too different from the template. Next, the actual matching is done by using a matching algorithm. Two different matching algorithms are tried here. One of them only accounts for translation shifts, while the other accounts for affine matches. The matching divides the word images into equivalence classes - each class presumably containing other instances of the same word.

6. Indexing may be done as follows. For each equivalence class, the number of elements in it is counted. The top n equivalence classes are then determined from this list. The equivalence classes with the highest number of words (elements) are likely to be stopwords (i.e. conjunctions like 'and' , articles like 'the', and prepositions like 'of') and may therefore be eliminated from further consideration. Let us assume that of the top n, m are left after the stopwords have been eliminated. The user then displays one member of each of these m equivalence classes and assigns their ASCII interpretation. These m words can now be indexed anywhere they appear in the document.

   In the experiments performed here, our main interest was in determining the feasibilty of the matching phase and hence the indexing phase was not carried out.

We will now discuss the matching techniques in detail.

## 7.4 Determination of Equivalence Classes

The list of words to be matched is first pruned using the areas and aspect ratios of the word images. The pruned list of words is then matched using a matching algorithm.

### 7.4.1 Pruning

It is assumed that

$$\frac{1}{\alpha} \leq \frac{A_{word}}{A_{template}} \leq \alpha \tag{7.1}$$

where $A_{template}$ is the area of the template and $A_{word}$ is the area of the word to be matched. The value of $\alpha$ used in the experiments was 1.3. A similar filtering step is performed using aspect ratios (ie. the width/height ratio). It is assumed that

$$\frac{1}{\beta} \leq \frac{Aspect_{word}}{Aspect_{template}} \leq \beta. \tag{7.2}$$

The value of $\beta$ used in the experiments was 1.7. In both the above equations, the exact factors are not important, but it should not be so large so that valid words are omitted, nor so small so that too many words are passed onto the matching phase. For more details on how the values for the area and aspect ratios are selected, the reader is referred to [42].

The use of the two matching algorithms SLH and DGauArea will now be briefly discussed.

#### 7.4.1.1 SLH Algorithm for Matching

Two sets of points I and J are created as follows. Let the word be composed of white pixels and the background be made of black pixels. Every white pixel in the first image is a member of the set I. Similarly, every white pixel in the second image is a member of set J. First, the centroids of the point sets are computed and the origins of each coordinate system is set at the respective centroid. The SLH algorithm (see Chapter 2) is now applied to these point sets and a correspondence between the two point sets is recovered. Note that some points may have no correspondence.

Given the correspondence between point sets I and J, the affine transform $(\mathbf{A}, \mathbf{t})$ can be determined by minimizing the following least mean squares criterion:

$$E_{SLH} = \sum_{l} (I_l - \mathbf{A}J_l - \mathbf{t})^2 \tag{7.3}$$

where $I_l$, $J_l$ are the (x,y) coordinates of point $I_l$ and $J_l$ respectively.

The values are then plugged back into the above equation to compute the error $E_{SLH}$. The error $E_{SLH}$ is an estimate of how dissimilar two words are and the words can, therefore, be ranked accordingly.

It will be assumed that the variation for valid words is not too large. This implies that if $a_{11}$ and $a_{22}$ are considerably different from 1, the word is probably not a valid match. If either of $a_{11}$ or $a_{22}$ is less than 0.8 or greater than 1/0.8, that word is eliminated from the rankings.

Note: The SLH algorithm assumes that pruning on the basis of the area and aspect ratio thresholds is performed.

#### 7.4.1.2 DGauArea

The DGauArea algorithm assumes greyscale images. Running the DGauArea algorithm directly on the input greyscale images did not work. This occurs because of noise, ink fading and other problems with the original data. The binary image created by thresholding has the words emphasized while eliminating noise artifacts. To apply DGauArea on this image requires that the binary images be converted into greylevel images by rescaling their intensities to the range (0-255). Otherwise, the DGauArea algorithm is the same as described in Chapter 5. To apply the DGauArea algorithm, an initial translation estimate is computed using the centroids of the images. The residual error is used to rank the words.

## 7.5 Experiments

The two matching techniques were tested on two handwritten pages, each written by a different writer. The first page (Figure 7.2) can be obtained from the DIMUND document server on the internet http://documents.cfar.umd.edu/resources/database/ handwriting.database.html. This page will be referred to as the Senior document. The handwriting on this page is fairly neat (see Figure 7.2). The second page is from an actual archival collection - the Hudson collection from the library of the University of Massachusetts (part of the page is shown in Figure (7.3). This page is part of a letter written by James S. Gibbons to Erasmus Darwin Hudson in 1842. The handwriting on this page is difficult to read and the indexing technique helped in deciphering some of the words.

Both pages were segmented into words (see [45] for details). The algorithm was then run on the segmented words. In the following figures, the first word shown is the template. After the template, the other words are ranked according to the match error. Note that only

the first few results of the matching are shown although *the template has been matched with every word on the page*.

### 7.5.1 Experiments Using the SLH Algorithm

Experiments were performed on both the Senior and Hudson documents. A few examples are shown here (for more details see [43]). The value of $\sigma$ depends on the expected translation; since it is small, $\sigma = 2.0$. A lower value of $\sigma = 1.5$ yielded poorer results.

The matches for the template "Lloyd" are shown in Figure (7.4). The entries are ranked according to the match error $E_{SLH}$. There are five instances of the word "Lloyd" in the document and they are ranked ahead of all other words.


**Figure 7.4.** Rankings for template "Lloyd" for the SLH algorithm.

The algorithm was also tested on the Hudson document (Figure (7.3)). An example of this ranking is shown below. Figure 7.5 shows the first 5 matches. There are two instances of the word "Standard" in this document (the first and third images in Figure 7.5). The transformation between the two instances is not affine and the word is, therefore, a difficult one to deal with. Notice that the two instances of "Standard" look somewhat different. The second instance (the third image in Figure 7.5) has spaces between the "t" and the "a" and also between the "d" and the "a" which are missing in the first image. In spite of that, the second instance of "Standard" is ranked 3rd in the document (there are about 300 words in this document).

### 7.5.2 Experiments Using the DGauArea Algorithm

The performance of DGauArea on the same words is displayed here.


**Figure 7.5.** Rankings for template "Standard" for the SLH algorithm.

The rankings for the template "Lloyd" from the Senior document are shown in Figure 7.6. The performance is similar to that of SLH.


**Figure 7.6.** Rankings for template "Lloyd" for the DGauArea algorithm.

The DGauArea algorithm's ranking when the word "Standard" is used as a template is shown in Figure 7.7. As discussed in the previous section, the transformation between the two instances of "Standard" is not affine. Inspite of that the second "Standard" is ranked 3rd (as for the SLH algorithm).


**Figure 7.7.** Rankings for template "Standard" for the DGauAreaalgorithm (the rankings are ordered from left to right).

### 7.5.3 Recall–Precision Results

Recall–Precision results are now shown for both the SLH and the DGauArea algorithms on both documents. Recall is defined as the "proportion of relevant documents actually retrieved" while precision is defined as the "proportion of retrieved documents that are relevant" [72]. Figure 7.8 shows the recall–precision results for both algorithms on the

Hudson document. The average precision for DGauArea and SLH on the Hudson document is 87.5 % and 86.6 % respectively. On the Senior document, the average precision for DGauArea and SLH is 93.1 % and 94.4 % respectively 7.9. We believe, that the difference between the values is statistically significant. DGauArea performs slightly worse than SLH on the Senior document but performs slightly better on the Hudson document. Note that the Hudson document is difficult for even people to decipher.

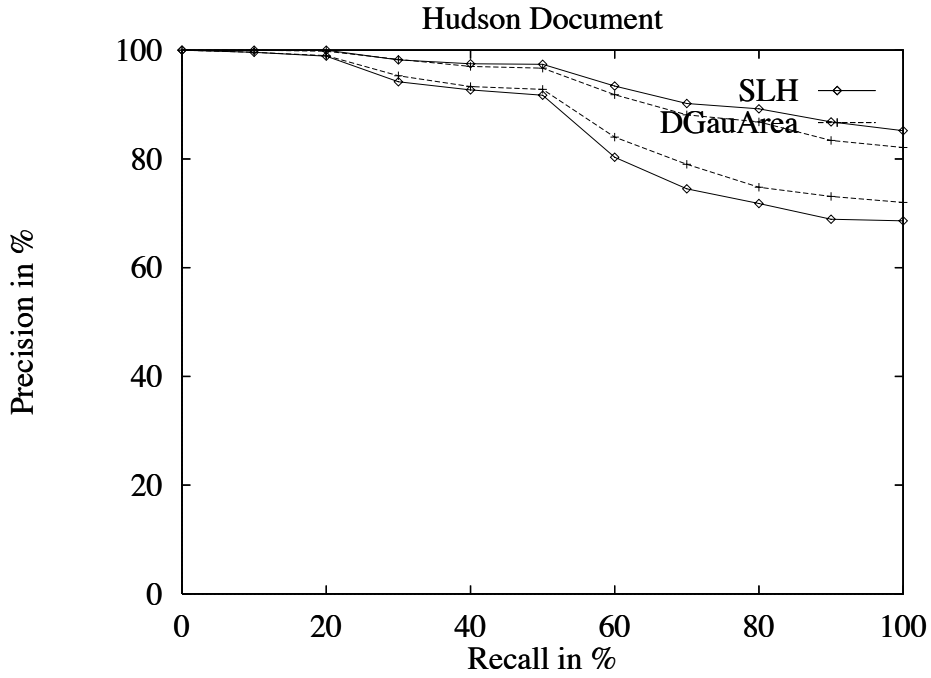



**Figure 7.8.** Recall–Precision for the SLH and DGauArea algorithms on the Hudson document.

### 7.5.4 Comparison of DGauArea and SLH

DGauArea also has other advantages compared to SLH. A non-optimized version of it takes about 1 hr/page compared to about 5.5 hrs/page for SLH. There is considerable room for improving the speed of the DGauArea algorithm applied to this problem, while it is doubtful that the speed of SLH can be significantly improved. In addition, DGauArea is more stable than SLH. The latter requires the computation of the singular value decomposition of large (200 by 200) matrices which are often poorly conditioned.

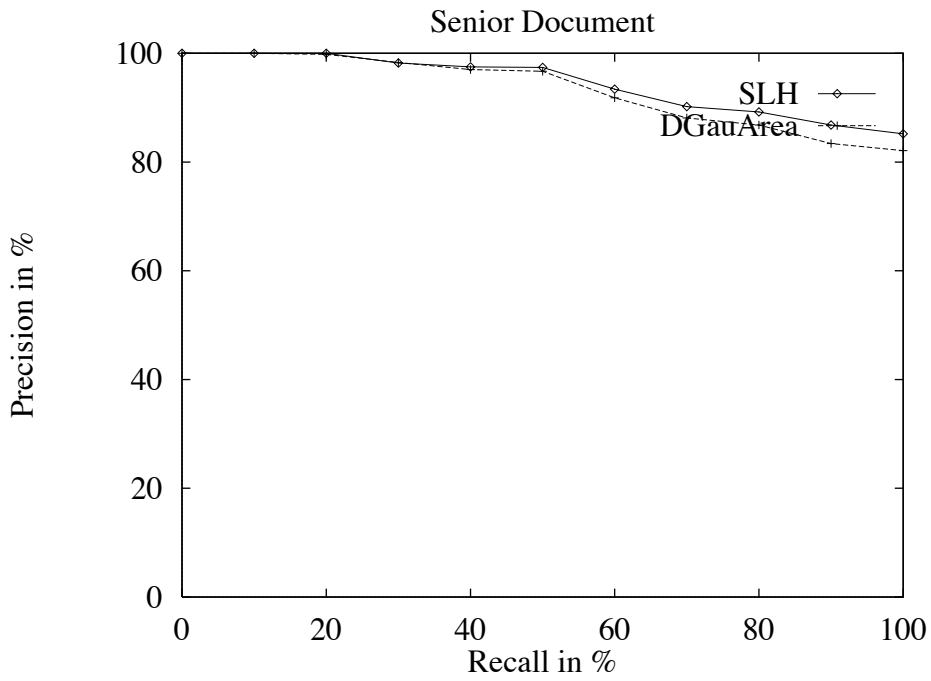



**Figure 7.9.** Recall–Precision for the SLH and DGauArea algorithms on the Senior document.

Currently, DGauArea computes an error measure over all points in both images. For the problem of word spotting, two images (obtained by taking different instances of the same word) may have many points which are not common to the two images. It may be desirable to detect and eliminate outliers to improve the error measure and hence the ranking of DGauArea.

# CHAPTER 8

# POINTS AND LINES

The discussion in the previous chapters centered on using image features for recovering affine transforms. The same framework may be extended to recovering affine transforms from point and line tokens. That is, given a set of points in one image and an affine transformed version in the second, the affine transform may be recovered. The main advantage of this technique is that explicit point-wise correspondence is not required; this is automatically obtained while measuring the affine transform. Since a set of points does not represent a continuous function, the actual equations and expressions will be slightly different than for brightnesses. For this method to work satisfactorily, a large number of points must be available otherwise incorrect solutions will be obtained. Further, no occlusion may occur (this may be relaxed by picking the largest subset of non-occluded points).

In this chapter, the same framework will be used to recover affine transforms and correspondences for points and lines. Consider the following equation for brightnesses first derived in Chapter 3.

$$\int F_1(\mathbf{r})d\mathbf{r} = F_2(\mathbf{r}_1)d\mathbf{r}_1 det(A^{-1}) \tag{8.1}$$

This equation needs to be modified when points are used because there is a difference between integrating a continuous function and a discrete function. To simplify matters, consider the 1D case. Let there be a function $f(x) = 1$ (see Figure 8.1a). Now if the function is bounded from $x_1$ to $x_k$ then:

$$\int_{-\infty}^{\infty} f(x)dx = \int_{x_1}^{x_k} f(x)dx = \alpha \tag{8.2}$$

where $\alpha$ is the area under the function. Consider also the discrete valued function $u = u(x_i) = 1$ for $i = 1, 2, .., k$ (see Figure 8.2a). In this case, the integration is replaced by a summation process. Thus,

$$\sum_{i=i}^{i=k} u(x_i) = k \tag{8.3}$$

Now let there be a scale change s so that

$$x' = sx \tag{8.4}$$

where $x'$ is the new coordinate. Then consider the function $f(x') = f(sx) = 1$ (Figure 8.1b) for all $sx_1 \leq x' \leq sx_k$ and

$$\int_{-\infty}^{\infty} f(x')dx' = \int_{sx_1}^{sx_k} f(sx)d(sx) = s\alpha \tag{8.5}$$

However, for the discrete function (Figure 8.2b) the sum remains the same under the scale change (since the sum is exactly the number of points).

$$\sum_{i=i}^{i=k} u(sx_i) = k \tag{8.6}$$

Thus, under a scale change the discrete and continuous versions behave differently. Therefore, the algorithms need to be modified to account for the difference. In 2D the discrete and continuous versions are represented by point tokens and (continuous) images respectively. Line tokens behave like point tokens perpendicular to the line and like images along the line.

## 8.1 Points

A set of points may be represented using delta functions. i.e

$$F_1(\mathbf{r}) = \sum_i \delta(\mathbf{r} - \mathbf{b}_i) \tag{8.7}$$

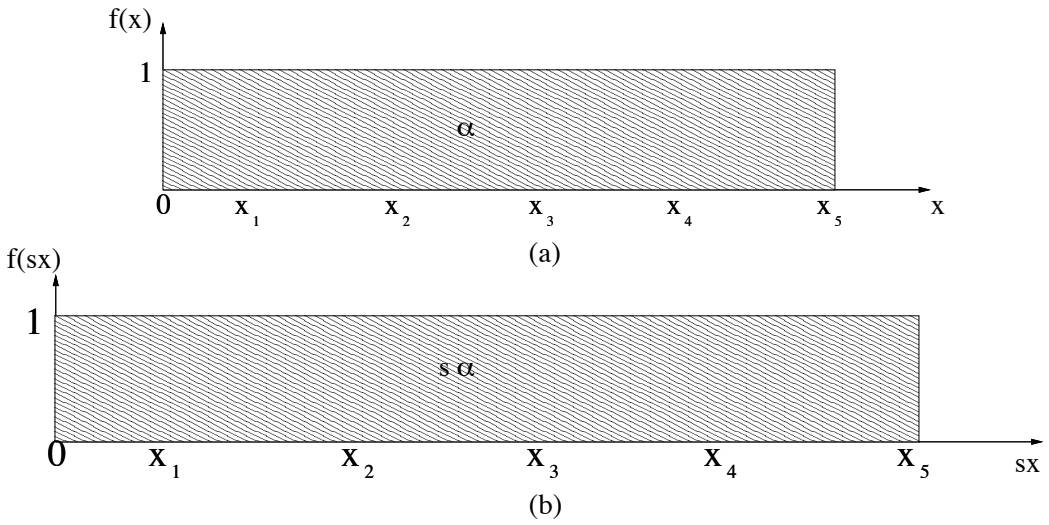



**Figure 8.1.** A continuous function.

The affine transformed version will therefore be

$$F_2(\mathbf{r}) = \sum_i \delta(\mathbf{Ar} + \mathbf{t} - \mathbf{b}_i) \tag{8.8}$$

Again, translation can be assumed to be zero. It can be recovered as discussed before. For points, translation can also be recovered by matching the centroids of the set of points defining $F_1$ and $F_2$.

As discussed in chapter 3, it is desirable to filter $F_1$ and $F_2$. Since $F_1$ and $F_2$ are discrete functions, they are not Riemann integrable. Instead the integrals must be interpreted as Stieltjes integrals. Then it may be shown that if un-normalized Gaussians $H(\mathbf{r}, \sigma^2\mathbf{I})$ are used, equality is again obtained between the filter outputs of $F_1$ and $F_2$. i.e.

$$\int F_1(\mathbf{r}) H(\mathbf{r}, \sigma^2\mathbf{I}) d\mathbf{r} = \int F_2(\mathbf{Ar}) H(\mathbf{Ar}, \mathbf{R\Sigma R}^T) d(\mathbf{Ar}) \tag{8.9}$$

where $H$ is a an un-normalized generalized (elliptical) Gaussian defined by $H(\mathbf{r}, \mathbf{M}) = exp(-\mathbf{r}^T\mathbf{M}^{-1}\mathbf{r}/2)$ and $\mathbf{M}$ is a symmetric positive semi-definite matrix.

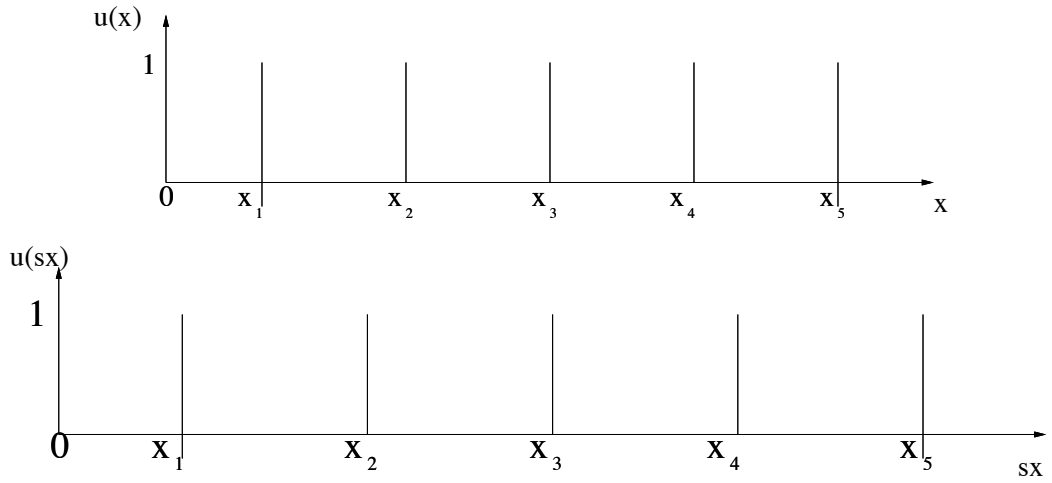



**Figure 8.2.** A discrete set of points.

### 8.1.1 Linearization for the Points Case

The solution of the above equation, though slightly different from the brightness case, may be obtained in the same way as for the brightness case. All the considerations that apply to the brightness case - the use of multiple scales and the use of different operating points - also apply here. These considerations will therefore not be discussed here. The linearized equation for the similarity and general affine cases are derived below.

#### 8.1.1.1 Case $\mathbf{A} = s\mathbf{R}(\theta)$

As in chapter 4, by linearizing with respect to $\sigma$, the solution for the similarity case with known translation may be shown to be:

$$F_1 * H(.,\sigma) \approx F_2 * H(.,\sigma) + (s-1)\sigma^2 F_2 * [\nabla^2 H(.,\sigma) + nH(.,\sigma)/\sigma^2] \qquad (8.10)$$

where n is the number of dimensions. Note the extra term as compared to the brightness equation (4.4).

#### 8.1.1.2 General Affine Transforms

Under a general affine transform, the Gaussian equation may be written as:

$$F_1(\mathbf{r}) * H(\mathbf{r}, \sigma) \approx F_2(\mathbf{Ar}) H(\mathbf{Ar}, \sigma^2 \mathbf{A}\mathbf{A}^T) \tag{8.11}$$

where the H on the right is an elliptical (un-normalized) Gaussian.

Linearized equations may be derived by following the same procedure in Section 5.1. The main difference is that instead of equation 5.5 we have:

$$H(\mathbf{A}^{-1}\mathbf{r}_1, \sigma) = H(\mathbf{r}_1, \sigma^2 \mathbf{A}\mathbf{A}^T) \tag{8.12}$$

Unlike equation 5.5 there is no determinant term in this equation. Using this relationshop, equation 8.11 may be rewritten as:

$$F_1(\mathbf{r}) * H(\mathbf{r}, \sigma) = F_2(\mathbf{r}_1) * H(\mathbf{A}^{-1}(\mathbf{r}_1 - \mathbf{t}), \sigma) \tag{8.13}$$

The linearization procedure and the identities in Section 5.1 all hold if G is replaced by H. Thus, the linearized version of H (obtained by substituting H for G in 5.12) is:

$$\begin{aligned} H(\mathbf{A}^{-1}(\mathbf{r}_1 - \mathbf{t}), \sigma) \;\approx\; & H(\mathbf{r}_1, \sigma) + \sigma^2[(b_{11} H_{x_1 x_1}(\mathbf{r}_1, \sigma) \\ & + \; b_{12} H x_1 y_1(\mathbf{r}_1, \sigma) \\ & + \; (b_{21} H_{x_1 y_1}(\mathbf{r}_1, \sigma) \\ & + \; b_{22} H_{y_1 y_1}(\mathbf{r}_1, \sigma)] \\ & - \; t_x H_{x_1}(\mathbf{r}_1, \sigma) - t_y H_{y_1}(\mathbf{r}_1, \sigma) \\ & + \; (b_{11} + b_{22}) H(\mathbf{r}_1, \sigma). \end{aligned} \tag{8.14}$$

Substituting this expression in equation 8.13 gives:

$$\begin{aligned} F_1(\mathbf{r}) * H(\mathbf{r}, \sigma) - F_2(\mathbf{r}_1) * H(\mathbf{r}_1, \sigma) \;\approx\; & b_{11}[\sigma^2 F_2 * H_{x_1 x_1}(\mathbf{r}_1, \sigma) + F_2 * H(\mathbf{r}_1, \sigma)] \\ & + \; b_{12} \sigma^2 F_2 * H_{x_1 y_1}(\mathbf{r}_1, \sigma) \end{aligned}$$

$$
\begin{aligned}
&+ \quad (b_{21}\sigma^2 F_2 * H_{x_1y_1}(\mathbf{r}_1,\sigma) \\
&+ \quad b_{22}[\sigma^2 F_2 * H_{y_1y_1}(\mathbf{r}_1,\sigma) + H(\mathbf{r}_1,\sigma)] \\
&- \quad t_x F_2 * H_{x_1}(\mathbf{r}_1,\sigma) - t_y F_2 * H_{y_1}(\mathbf{r}_1,\sigma) \qquad (8.15)
\end{aligned}
$$

Note the extra terms $b_{11}H(\mathbf{r}_1,\sigma)$ and $b_{22}H(\mathbf{r}_1,\sigma)$ as compared to equation 5.13.

If the filtering is done not only at the origin but at a point $\mathbf{l}_i$, then

$$
\begin{aligned}
F_1(\mathbf{r}) * H(\mathbf{r},\sigma) - F_2(\mathbf{r}_1) * H(\mathbf{r}_1,\sigma) \quad &\approx \quad b_{11}[\sigma^2 F_2 * H_{x_1x_1}(\mathbf{r}_1,\sigma) + F_2 * H(\mathbf{r}_1,\sigma)] \\
&+ \quad b_{12}\sigma^2 F_2 * H_{x_1y_1}(\mathbf{r}_1,\sigma) + b_{21}\sigma^2 F_2 * H_{x_1y_1}(\mathbf{r}_1,\sigma) \\
&+ \quad b_{22}[\sigma^2 F_2 * H_{y_1y_1}(\mathbf{r}_1,\sigma) + H(\mathbf{r}_1,\sigma)] \\
&- \quad t_x F_2 * H_{x_1}(\mathbf{r}_1,\sigma) - t_y F_2 * H_{y_1}(\mathbf{r}_1,\sigma) \\
&- \quad (\mathbf{B}\mathbf{l}_i)^T H'(\mathbf{r}_1,\sigma) \qquad (8.16)
\end{aligned}
$$

where the $\mathbf{H}'$ denotes the first derivative of $H$ i.e $H_{x_1}, H_{y_1}$.

It is straightforward to use this equation to derive an algorithm for points similar to GauArea for image features. Derivative of Gaussian equations similar to those for image features may also be derived.

### 8.1.2 Experiments

To demonstrate that the theory derived above is sound, an experiment was conducted to show that the algorithm for the point tokens retrieves the correct affine transform and correspondence.

Six point tokens were input to the algorithm. The (x,y) coordinates of the point tokens were (1,2),(-3,6),(5,-4),(-7,-8),(4,3),(5,6). The points were synthetically affine transformed using the following affine transformation $\mathbf{A} = \begin{pmatrix} 1.1 & 0.2 \\ 0.2 & 1.1 \end{pmatrix}, \mathbf{t} = (0,0)$. The correct affine transform was recovered in 3 iterations.

The experiment was repeated after adding random noise to the point coordinates. The noise was uniformly distributed between 0 and 0.5. The recovered affine transform was $\mathbf{A} = \begin{pmatrix} 1.11 & 0.19 \\ 0.21 & 1.09 \end{pmatrix}, \mathbf{t} = (0.27, 0.27)$.

Thus, it is clear that the algorithm works. Versions of the algorithm which do not take into account the deformation, i.e. algorithms for points similar to GauAreaN for images ( see chapter 5), do not converge. The algorithms for recovering the affine transform and correspondence from point tokens is likely to be sensitive to occlusions and missing points. This will not be investigated here.

## 8.2 Lines

Equations may also be derived for the case of lines. The advantage of using this framework for lines is that the method can deal with both closed and open curves as well as straight and curved lines. The method will also work on a collection of line segments. No correspondence is required, although it is assumed that if line segments are used, the same segments are used in both images.

In the case of lines, Riemann integration must be performed along the line and Stieltjes integration perpendicular to the line. This makes the equations somewhat messy for the case of the general affine since the local line orientation must be factored in. However, for the similarity case, the local line orientation does not enter into the equations. In this case, the Gaussian filter equation may be shown to be

$$F_1 * H(., \sigma)/\sigma = F_2 * H(., s\sigma)/(s\sigma) \tag{8.17}$$

where it is assumed that $F_1$ and $F_2$ are defined in 2-D dimensions (the general case is a straightforward extension).

The point, line and brightness cases may now be contrasted. In the point case (equation (8.10)), both the $\sigma's$ required for normalizing a 2-D Gaussian are absent ; in the line case only one is absent (equation (8.17)), while in the brightness case both are present.

The solution for lines in the similarity case is again obtained by linearizing with respect to $\sigma$ and is given by

$$F_1 * H(.,\sigma) \approx F_2 * H(.,\sigma) + (s-1)\sigma^2 F_2 * [\nabla^2 H(.,\sigma) + H(.,\sigma)/\sigma] \tag{8.18}$$

For the method to work well with lines, it is important to localize lines with sub-pixel accuracy.

The general case for lines is messy since it depends on the local orientation of the line segment. It will not be considered further here.

## 8.3 Comments on Points and Lines

Both points and lines are for the most part unaffected by illumination changes and shading. So if this is a significant concern, they can be used. A number of man-made scenes often consist of homogeneous regions surrounded by lines. In such situations where there is minimal image texture, methods using image brightnesses will fail. However, line based methods may still work. Note that filter sizes may need to be changed depending on the size of the structures present in the image.

### 8.3.1 Combining Points, Lines and Brightnesses

One advantage of this framework is that it provides a natural mechanism to combine points, lines and brightnesses (just put them all in one big matrix and use singular value decomposition). For example, in regions in the images where there are strong boundaries and corner points, they are weighted more heavily by using this technique.

# CHAPTER 9

# CONCLUSIONS

In this dissertation, the problem of matching affine-transformed images has been investigated. Two image patches differing by an affine transformation are different in size and shape; i.e. under an affine transformation an image patch undergoes a geometric distortion.

It was shown that this geometric distortion must be accounted for when two affine-transformed images are matched, particularly if the affine transform is large. If the images are filtered with Gaussians and the filter outputs are to be equal, then if the first image patch is filtered with a Gaussian, the second must be filtered with a deformed Gaussian; the deformation being precisely the affine transformation between the images. Similar constraints hold when the image is filtered with derivatives of Gaussians.

The affine transform may be solved for by finding the deformation of the Gaussian such that the two filter outputs are equal. This exhaustive approach is computationally expensive. Instead, an alternative is proposed here, which consists of coarsely sampling the Gaussian filter. Between samples, the Gaussian is linearized to solve for the affine transform. This linearization takes the deformation into account.

It was shown that a set of linearized Gaussian and Gaussian derivative filters at a single point in the image produced poor results. Instead by pooling the linearized filter outputs from a number of points, iteratively refining the solution and using multiple scales, robust algorithms may be produced. Two such algorithms are created. The first, GauArea uses linearized Gaussian filters while the second, DGauArea uses linearized first derivative of Gaussian filters.

Experiments on both synthetic and real images showed that GauArea and DGauArea performed better than versions of the algorithms which did not account for filter deformation. The difference is particularly noticeable for the experiments on real images. Large affine transforms may be solved using these algorithms. It is shown that scale changes up to about 2.5, in-plane rotations up to 35 degrees and shear deformations up to 0.8 may be solved for using GauArea and DGauArea. The experiments showed that for real images, moderate window sizes (roughly 24 by 24 pixels) are desirable and that the filter widths must be at least 8 times the standard deviation of the filters.

The algorithms can recover translations of a few pixels. For larger translations, an initial estimate must be provided. This estimate can be derived in different ways which depend on the problem to be solved. Depending on the task, this estimates may be derived by by comparing the centroids of images, by sampling or by using a coarse–to–fine strategy. In Section B, a strategy to find large translations by comparing filter ouputs is suggested.

## 9.1 Future Work

Any complex study often creates more questions than it answers. This dissertation leaves some of these questions for the future.

In Section B, a technique for computing gross translations was suggested. A similar technique has also been applied to the problem of image retrieval [47]. However, there are certain unresolved issues with the use of this technique. For example, the choice of scales is an open issue. It is important to pick scales in both images which do not undergo scale space transitions.

In the experiments, constraint equations from every point point within a window were used. However, it is not necessary to obtain linearized equations at every point within a window - for example, points in homogeneous regions may not contribute any information. From a computational standpoint, it is cheaper to use fewer points. It would be useful to derive measures which show which points may be omitted and which should be included.

The algorithms suggested here used least means squares to solve the problem. It may be desirable to use robust estimators instead in some of the situations. For example, the word spotting application may benefit from estimators which reject outlier points.

Point based, line based and image based algorithms were suggested by the framework used in this study. There is a need for more experimentation with the point based techniques. It may also be useful to combine point and image based techniques for the same pair of images. Depending on whether token points or image features are more reliable in a particular context, the appropriate entity may be weighted more heavily.

# APPENDIX A

# DERIVATION OF RECURRENCE RELATIONS

In this appendix the recurrence relation used in chapter 5 will be derived. It is easiest to do this in the Fourier domain.

In the following discussion

$$g(x) \leftrightarrow G(f) \tag{A.1}$$

denotes a Fourier transform pair.

We start by noting that the following form of the Gaussian is its own Fourier transform. ([19]):

$$\exp(-\pi x^2) \leftrightarrow \exp(-\pi f^2) \tag{A.2}$$

The Fourier transform of a normalized Gaussian is, therefore, given by

$$G(x,\sigma) \equiv \frac{1}{\sqrt{2\pi}\sigma} \exp(-x^2/2\sigma^2) \leftrightarrow \exp(-2\pi f^2\sigma^2) \equiv G(f,\sigma) \tag{A.3}$$

In 2D the Fourier transform of a normalized Gaussian may, therefore, be written as

$$G(x,y,\sigma) \equiv \frac{1}{2\pi\sigma} \exp[-(x^2+y^2)/(2\sigma^2)] \leftrightarrow \exp[-2\pi(f_x^2+f_y^2)\sigma^2] \equiv G(f_x,f_y,\sigma) \tag{A.4}$$

Using the derivative property of Fourier transforms gives ([19])

$$\frac{d^n}{dx^{n-m}dy^m} G(x,y,\sigma) \leftrightarrow (j2\pi f_x)^{n-m}(j2\pi f_y)^m G(f_x,f_y,\sigma) \tag{A.5}$$

Differentiating the right hand side of the above equation with respect to $f_x$, therefore, gives the following Fourier transform pair:

$$-j2\pi x\frac{d^n}{dx^{n-m}dy^m}G(x,y,\sigma) \leftrightarrow d/df_x[(j2\pi f_x)^{n-m}(j2\pi f_y)^m G(f_x,f_y,\sigma)] \tag{A.6}$$

The right hand side may be expanded as follows.

$$\begin{aligned} & d/df_x[(j2\pi f_x)^{n-m}(j2\pi f_y)^m G(f_x,f_y,\sigma)] \\ = \;& j2\pi(n-m)(j2\pi f_x)^{n-m-1}(j2\pi f_y)^m G(f_x,f_y,\sigma) \\ + \;& (j2\pi f_x)^{n-m}(j2\pi f_y)^m(-4\pi^2\sigma^2 f_x)G(f_x,f_y,\sigma)] \end{aligned} \tag{A.7}$$

The inverse Fourier transform of the right hand side of equation A.7 is equal to

$$j2\pi(n-m)\frac{d^{n-1}}{dx^{n-m-1}dy^m}G(x,y,\sigma)+\sigma^2\frac{d^{n+1}}{dx^{n-m+1}dy^m}G(x,y,\sigma) \tag{A.8}$$

Using equation A.6 and simplifying gives the following recurrence equation

$$x\frac{d^n}{dx^{n-m}dy^m}G(x,y,\sigma) = -(n-m)\frac{d^{n-1}}{dx^{n-m-1}dy^m}G(x,y,\sigma)-\sigma^2\frac{d^{n+1}}{dx^{n-m+1}dy^m}G(x,y,\sigma) \tag{A.9}$$

In terms of the quantity $\phi$ defined in chapter 4, the recurrence relation may be written as:

$$x\phi^{n-m,m} = -\sigma(n-m)\phi^{n-m-1,m}-\sigma\phi^{n-m+1,m} \tag{A.10}$$

A similar equation may be derived in the y direction:

$$y\phi^{n-m,m} = -\sigma m\phi^{n-m,m-1}-\sigma\phi^{n-m,m+1} \tag{A.11}$$

# APPENDIX B

# RECOVERING GROSS TRANSLATIONS

The approach adopted in Chapter 5 to recover affine transforms works for translations of a few pixels. It does not work if larger translations are involved. This is due to two reasons:

1. The support of the Gaussian derivatives determines the extent over which a linearized estimate is valid.

2. Linearization is predicated on the assumption that the correct results are close to the point about which linearization is performed.

Affine transforms with larger translations can be solved by providing an initial estimate of the translation. This may be obtained by using one of several techniques:

1. Sampling the space of translations.

2. Using a pyramid.

3. Comparing the Gaussian derivative filter outputs (or equivalently differential invariants) at corresponding points and requiring that they be similar.

Of these three techniques, only the last will be discussed in detail.

## B.1 Sampling the Space of Translations

A straightforward way to increase the range of translations that can be recovered is to sample the space of translations. Consider the Gaussian constraint equation

$$F_1 * G(\mathbf{r}, \sigma) = F_2 * G(\mathbf{A}\mathbf{r} + \mathbf{t}, \sigma) \tag{B.1}$$

This may be rewritten as:

$$F_1 * G(\mathbf{r} - \mathbf{t}_0, \sigma) = F_2 * G(\mathbf{A}\mathbf{r} + \mathbf{t}_r, \sigma) \tag{B.2}$$

where $\mathbf{t}_0$ is recovered by sampling and $\mathbf{t}_r$ is the residual left after sampling.

The main problem with sampling is that it gets expensive if the translations involved are even moderately large. Let the maximum possible translation along the x or y axis be k. Then $k^2$ possible translation samples need to be examined. If k is small (say $k \leq 5$) then it may be reasonable to use the sampling scheme. However, for k large the computation may be prohibitive.

## B.2 Using a Pyramid

Large translations may also be computed using a coarse–to–fine strategy (see Section 2.4 for a more detailed discussion). A pyramid is effective if the translational motion of a large portion of the image is being recovered. However, if the object in question occupies a small portion of the image, then there are difficulties in recovering the object's translation using a pyramid. It is also difficult using pyramids to recover large translations (e.g. if the object undergoes a motion which is $\geq 50\%$ of the image). It is also difficult to use a pyramid if the size of the object changes considerably - although it may be possible to handle such size changes by comparing different levels of the pyramid.

## B.3 Comparison of Gaussian Derivatives

Gross translation may also be recovered by comparing the outputs of images filtered with Gaussian derivatives at several scales. Consider a point $\mathbf{r}$ in the first image $F_1$ and its corresponding point $\mathbf{r}_1$ in the affine transformed second image $F_2$. The image intensities at these points and in their neighborhoods will be similar. If illumination, shading and shadowing are ignored, any difference will be due to geometric distortions caused by the affine transforms. Thus if the affine deformations are small, the difference between the filter outputs at the corresponding points will be small.

Kass [31] formulated a similar approach in the case of stereo where he computed disparity by assuming that a small value of the following error function indicates a correspondence.

$$\sum_x |F_1(x) * G_i(x,\sigma) - F_2(x') * G_i(x',\sigma)|^2 \tag{B.3}$$

where $G_i$ is the first derivative of a Gaussian. A small value of the error denotes correspondence only if the geometric distortion is small.

A rationale for this approach can be derived by considering the local jet [33] at a point. The local image irradiance pattern at a point can be expressed in the form of a Taylor series expansion of the brightness convolved with a set of Gaussian derivatives (the local jet) [33]. An approximation to the local jet may be obtained by taking the first few terms in the Taylor series. Two image irradiance patterns are the same if the corresponding terms in the Taylor series match with each other. The intensity may also vary due to changes in illumination. This can be avoided by considering only the filter outputs convolved with derivatives of Gaussians. Let the image irradiance pattern at two points $\mathbf{r}$ and $\mathbf{r'}$ be the same. Then at the points $\mathbf{r}$ and $\mathbf{r'}$ we have:

$$I_{x^i y^j} \equiv F(\mathbf{r}) * G_{x^i y^j}(.,\sigma) = F(\mathbf{r'}) * G_{x'^i y'^j}(.,\sigma) \tag{B.4}$$

where $G_{x^i y^j}(.,\sigma)$ denotes the energy normalized Gaussian differentiated i times with respect to x and j times with respect to y; the dependence of I on the image coordinates and $\sigma$ has been omitted for convenience.

Gaussian derivatives are sensitive to orientation. Orientation–independent estimates may be obtained by combining the Gaussian derivatives into expressions invariant to 2D rotation [16]. Up to order two, there are five 'irreducible' invariants (irreducible implies that no smaller set can be found).

$$\begin{aligned} d_0(\mathbf{r},\sigma) &= I && Intensity \\ d_1(\mathbf{r},\sigma) &= I_x^2 + I_y^2 && Magnitude \\ d_2(\mathbf{r},\sigma) &= I_{xx} + I_{yy} && Laplacian \\ d_3(\mathbf{r},\sigma) &= I_{xx}I_xI_x + 2I_{xy}I_xI_y + I_{yy}I_yI_y \\ d_4(\mathbf{r},\sigma) &= I_{xx}^2 + 2I_{xy}^2 + I_{yy}^2 \end{aligned} \tag{B.5}$$

A orientation–invariant vector $\mathbf{R}(\sigma)$ at a given scale may be constructed from the invariants. Since the first invariant is sensitive to greylevel shifts, it is omitted from this invariant vector. Thus

$$\mathbf{R}(\sigma)(\mathbf{r}) = [d_1(\mathbf{r},\sigma), d_2(\mathbf{r},\sigma), d_3(\mathbf{r},\sigma), d_4(\mathbf{r},\sigma)] \tag{B.6}$$

A multi-scale invariant vector $\Gamma(\mathbf{r})$ can be constructed by taking different $\sigma$'s. Thus

$$\Gamma(\mathbf{r}) = [\mathbf{R}(\sigma_1)(\mathbf{r}), \mathbf{R}(\sigma_2)(\mathbf{r}), , ..., \mathbf{R}(\sigma_n)(\mathbf{r})] \tag{B.7}$$

Schmid and Mohr [60] used a larger set of 9 invariants for object recognition. The matching phase of their system can be described as follows. They computed corner points in a set of images (which included a query image). They then found points in the set of

images which were similar to the corner points in the query image. Similarity was decided by comparing the first 9 invariants at a single scale using the Mahalanobis distance. Further pruning was done by requiring that the spatial configuration of the points in the images be similar to that in the query.

Schmid and Mohr's requirement that all 9 invariants match is overly restrictive and may not be satisfied in a number of cases. For example, repetitive textures where the texture elements change slighly may not satisfy this criterion (see Figure B.1(b)). Their criterion also may not be satisfied for images which differ by an affine transform. In their own work, they assumed that their images differed by a similarity transform. However, when the images are affine transformed with respect to each other, the invariants will not match exactly. The nth order invariant differs from its affine transformed version by a factor which is of the order of $det(\mathbf{A})^n$ where $\mathbf{A}$ is the affine deformation. The larger the invariant, the less likely it will be similar under affine changes. Thus, for general affine transforms it may be more difficult to assure that large order invariants do not change much under an affine transform.

It is, therefore, desirable to use lower order invariants. Using lower order invariants causes more false matches. The false matches may be reduced by using multiple scales. Such an algorithm to recover "similar" objects has been proposed in [47, 54]. The algorithm proposed here is similar in spirit to that in [47] but for the purpose of finding the possible correspondences for a given image feature point (for more detailed discussions see [47]).

Our algorithm uses only four invariants but at multiple scales. A multi-scale invariant vector $\Gamma(\mathbf{r})$ of a point in one image is compared with the multi-scale invariant vectors $\Gamma(\mathbf{r}')$ of all points in a neighboring image. If the difference is within a tolerance $Tol$ then the two points are hypothesized to be from similar patterns. There may be a number of points satisfying this criterion (either because the pattern is replicated) or because the criterion is insufficient to filter out all points. In either case, a verification phase is required.

The verification phase may be implemented using one of the affine transform algorithms discussed earlier in the chapter.

Let $\mathbf{r}$ be the image feature whose correspondence is being sought and $\mathbf{r'}$ be the image feature being compared. Then $\mathbf{r}$ and $\mathbf{r'}$ are hypothesized to have similar patterns if their multi-scale invariant vectors are similar - the comparison being done term by term i.e.

$$|\Gamma(\mathbf{r}) - \Gamma(\mathbf{r'})| < Tol \tag{B.8}$$

if $\forall i, |d_i(\mathbf{r}, \sigma) - d_i(\mathbf{r'}, \sigma)| \leq Tol$ where the $d_i$ are given by equation B.5.

The invariant vectors may be stored in a database and indexed by their value. If a binary tree is used to store the invariants, they may be searched for the correct matches in time proportional to log (number of pixels). For example, a 512 x 512 image which has $2^{18}$ pixels can be searched in time 18 K where K is some constant.

Unlike Schmid and Mohr [60], we do not use corner points. Corner points are advantageous because at certain scales many of the invariants have large values at those points. However, Schmid and Mohr found that their corner point detector was not stable over a large range of scales [60]. In addition, many situations require matching image features which may not be as well defined as corner points (see for example Figure B.4(a)). It is important that not all the invariants at a given image feature be small for correspondence to be properly detected. However, it is not necessary for the image structures (or features) used to have any geometric relationships with respect to the scene.

If a single scale is used, the number of hypothesized points is still quite large. Further pruning can be done by considering more filter scales. By using more scales, the constraint that the two patterns must be similar at more than one scale is imposed. However, if too many scales are used then no matching points may be found. By experimentation, it was found that two or three scales worked best. Two scales were used to generate the experiments in this section.

It is also desirable to ensure that the image features match even if a scale (size) change or an affine change occurs. Under a size change s, the Gaussian derivatives remain the same if energy normalized derivatives are considered. The energy normalized Gaussian derivatives may be defined as in equation 5.14 by:

$$\phi_{n-m,m}(\mathbf{r},\sigma) \equiv \sigma^n \frac{d^n G(\mathbf{r},\sigma)}{dx^{n-m}dy^m}. \tag{B.9}$$

Under a size change s

$$\phi_{n-m,m}(\mathbf{r},\sigma) = \phi_{n-m,m}(s\mathbf{r}, s\sigma) \tag{B.10}$$

Thus, if the invariants in equation B.5 are constructed from these energy normalized Gaussian derivatives, they will also be invariant under a size change (this is similar to the work done in Chapter 3 on similarity transforms).

It is not necessary to compute invariants for all possible values of s. A coarse sampling suffices. The value of the Gaussian derivatives and hence the invariants change, for the most part, slowly with scale. If the samples are chosen close enough, the value of the invariant at the sample scale will be close to the correct scale. Thus it suffices to compare the sample values. Let the invariants for the first feature be computed at $\sigma_1$ and let the invariants be sampled at $\sigma_1, \sigma_2, \sigma_3$. where $\sigma_3\sigma_2 = \sigma_2/\sigma_1$. Thus instead of making the comparison:

$$|d_i(\mathbf{r},\sigma_1) - d_i(\mathbf{r}', s\sigma)| < Tol \tag{B.11}$$

the following comparisons are made:

$$|d_i(\mathbf{r},\sigma_1) - d_i(\mathbf{r}', \sigma_j)| < Tol \tag{B.12}$$

where j = 1,2,3. Assume now that $s\sigma_1$ is closest to $\sigma_3$. Then in equation B.12, the values of the invariants are likely to be within the required tolerance. For practical purposes, it

suffices to use scales at steps of $\sqrt{(2)}$ i.e. adjacent samples differ by a factor of half an octave (these samples will be referred to as scale steps).

Affine transformations which are not similarity transforms can be handled by taking samples using elliptical Gaussians. For small and even moderate shears, the use of elliptical Gaussians can be avoided altogether and instead the nearest sampled filter with a circular Gaussian may be used.

Not all scales can be used in all situations. It is possible that the image feature may only have structure at a few scales. For example, some rapidly varying textures may only have information at high frequencies. For such images, it is important to use scales where the information is available. Again in certain situations, a scale space singularity may occur (for example two maxima may coalesce into a single maxima) [37]. Near such scales the invariants are not well behaved and it is necessary to avoid such scales. However, characterizing which scales to use and which to avoid is a difficult task and will not be done here.

### B.3.1 Experiments on Finding Gross Translations

A set of experiments will now be described. The first experiment compares the use of 9 invariants at a single scale (the procedure used by Schmid and Mohr [60]) with 4 invariants at two scales. Figure B.1(a) is a real image of a wall hanging. A point is marked on one of the repeated circular textures on the bottom of the wall hanging. The aim is to find all corresponding points in the same image - there are 9 similar structures in this image. Figure B.1(b) shows the output using 9 invariants at a single scale of 4 pixels. Notice that the same point has been recovered but all the similar texured structures on either side have been missed. It is clear that using 9 invariants is overly restrictive. Figure B.1(c) shows the output using 4 invariants at a two scales ($4$ and $4\sqrt{2}$ pixels). All 9 structures have been detected. The tradeoff compared to using 9 invariants at a single scale is the smearing (multiple matches) and the false match.

The next set of experiments investigates how the algorithm performs when affine transformed images are presented. Figure B.2(a) shows a photograph of a Pepsi can. The camera was moved and the Pepsi can again imaged from three different viewpoints (Figures B.2(b)–B.2(d)). Notice that the Pepsi can in Figure B.2(b) (referred to as Pepsi1) is roughly 1.4 times the initial image Figure B.2(a) (referred to as Pepsi0). The Pepsi can in Figure B.2(c) (referred to as Pepsi2) is almost twice as large as Pepsi0. The Pepsi can was also rotated by a large amount and scaled by a factor of roughly 1.4 and this is shown in Figure B.2(d) (referred to as Pepsi3).

Due to an artifact of the way digital images are taken, the scaling may be different in the x and y directions. For example, for the images in the pictures shown the ratio of the scales is approximately 0.7/1 (x/y). One consequence of this ratio not being equal to 1 is that a circle will be imaged as an ellipse even when viewed head on. A similar situation occurs in the two images Pepsi3 and Pepsi0 where the scaling along the two two perpendicular directions is different.

Three points are chosen on the Pepsi can - shown in images Figure B.2(a), Figure B.3(a) and Figure B.4(a). Image Pepsi0 is filtered at two scales ($\sigma_1 = 3$ pixels, $\sigma_2 = 3 * \sqrt{2}$ pixels) and a multi-scale invariant vector computed for the chosen points. The other figures are also filtered at a number of scales. The results for images Pepsi1 and Pepsi3 are shown after filtering at scales $\sqrt{2}$ larger while the result for Pepsi2 is shown after filtering at scales 2 times larger than those for Pepsi0 (these are the scales that would be expected given the affine deformations.) At other scales, fewer points are usually obtained and none of these are matches.

The multi-scale invariant vectors are then compared and a number of points hypothesized to be matches. Figures B.2(b)–B.2(d) illustrate the results for point 0, Figures B.3(b)–B.3(d) for point 2 and Figures B.4(b)–B.4(d) for point 3. Each figure also lists the number of hypothesized points that were found.

Notice that a number of points tend to clump together. Points close to a corresponding point are also often marked as matches, because they have invariant values similar to the correct match point. There are also some false matches which may be eliminated by running the affine algorithm.

Figure B.5(a) shows a point (point 1) on Pepsi0 which is not on the pepsi can. Figures B.5(b)–B.5(d) show the results of comparing the multi-scale invariant vectors for the corresponding images. Note that point 1 is on a structure whose size change is much smaller than the pepsi can. In fact, the size change of the structure depends on its position in the image (since it is not frontal). Thus there are points in the structure whose size changes are of the order of 1.1 over all images while other points have size changes of the order of 1.3 in some images (noticebly in Pepsi3). It is to be, therefore, expected that some points on this structure will be found by filtering at the same scale step while others will be found by filtering one scale step higher and this is what occurs in practice. Figures B.5(b) and B.5(d) are produced by using the same scale step while Figure B.5(c) is produced at one scale step higher. Notice that although there are a large number of points - many of these are valid since they correspond to features which are similar to the original feature in Pepsi0.

### B.3.2 Comments on Recovering Gross Translations

The technique described above recovers a set of possible correspondences for each point by comparing the values of invariants. In all the experiments described, the correct correspondence (or correspondences if there are multiple ones) is in the set of possible correspondences. The set of possible correspondences is small. However, this

(a) Initial image Krish with a point marked on it.

(b) Image Krish with hypothesized points using 9 invariants. Number of points is 1. Note that corresponding points in other nearly identical textures have all been missed.

(c) Image Krish with hypothesized points using our technique. Number of points is 49. Corresponding points have been located in all similar image structures. There is also a false match.

**Figure B.1.** Hypothesized corresponding points for point marked in Krish image.


(a) Initial image Pepsi0 with the point 0.


(b) Image Pepsi1.Number of points is 7.


(c) Image Pepsi2. Number of points is 18.


(d) Image Pepsi3. Number of points is 9.

**Figure B.2.** Hypothesized corresponding points for point 0.

(a) Initial image Pepsi0 with the point 2.

(b) Image Pepsi1. Number of points is 4.

(c) Image Pepsi2. Number of points is 12.

(d) Image Pepsi3. Number of points is 18.

**Figure B.3.** Hypothesized corresponding points for point 2.


(a) Initial image Pepsi0 with the point 3.


(b) Image Pepsi1.Number of points is 12.


(c) Image Pepsi2. Number of points is 6.

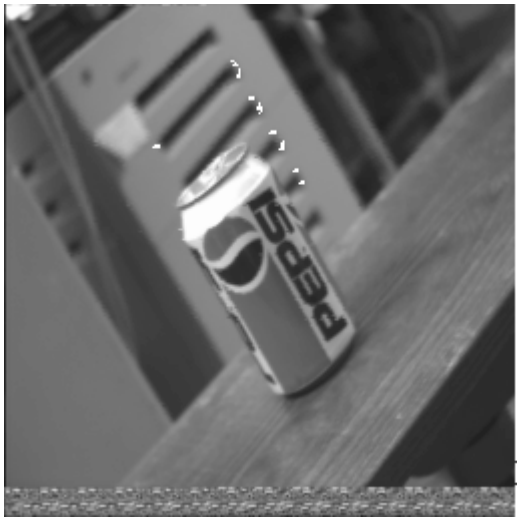

(d) Image Pepsi3. Number of points is 12.

**Figure B.4.** Hypothesized corresponding points for point 3.


(a) Initial image Pepsi0 with the point 1.


(b) Image Pepsi1.Number of points is 13.


(c) Image Pepsi2. Number of points is 14.



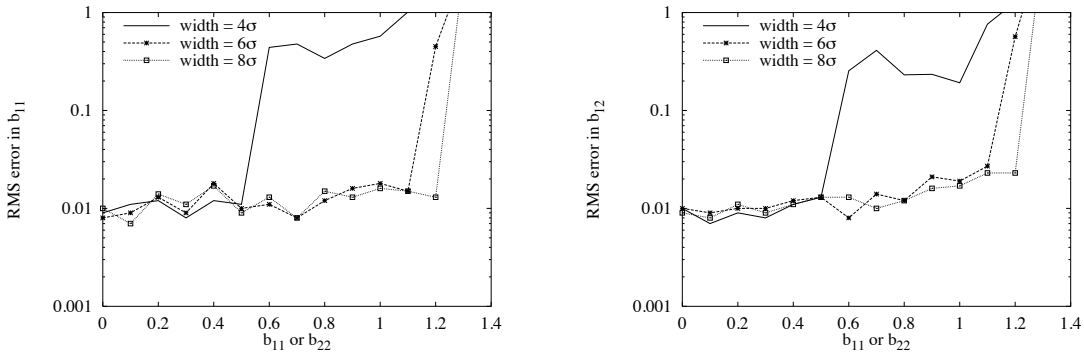

(d) Image Pepsi3. Number of points is 26.

**Figure B.5.** Hypothesized corresponding points for point 1.

# APPENDIX C

# ADDITIONAL GRAPHS

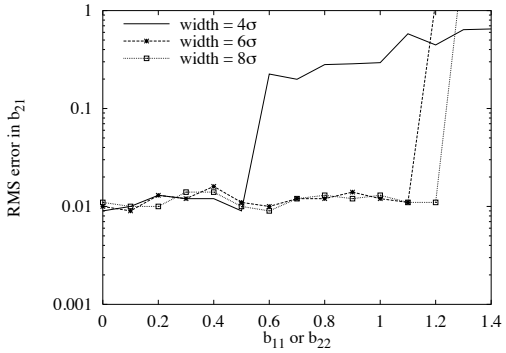
This Appendix contains additional plots for experiments conducted in Chapter 6. The reader is referred to Chapter 6 for details of these plots.



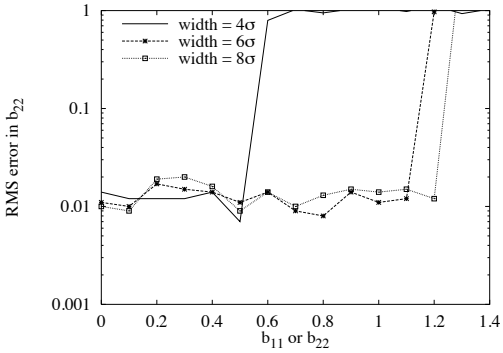

(a) RMS error in $b_{11}$



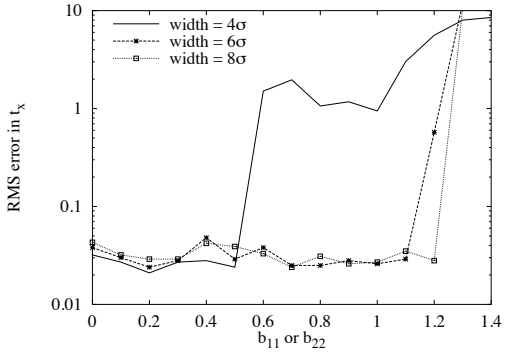

(b) RMS error in $b_{12}$



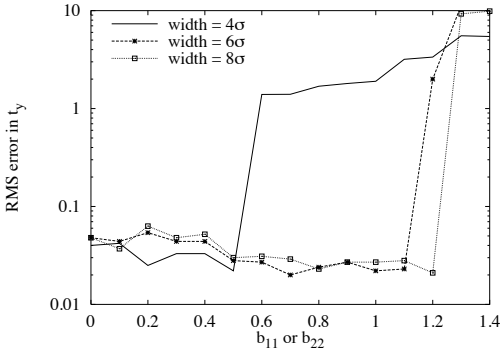

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

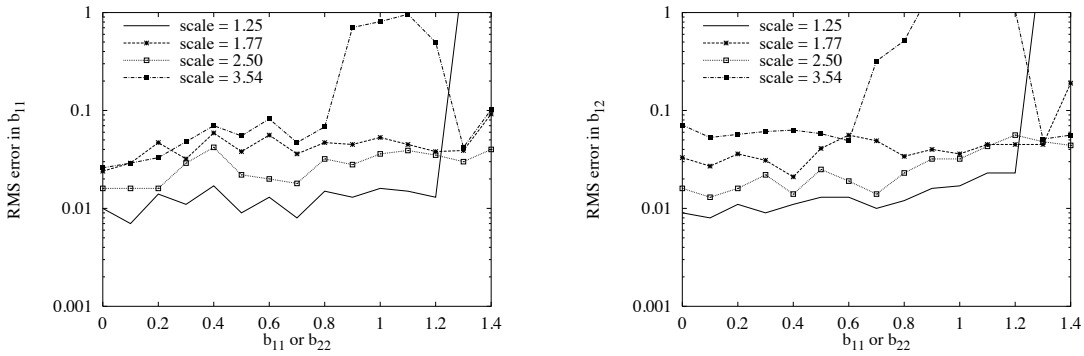



(e) RMS error in $t_x$




(f) RMS error in $t_y$

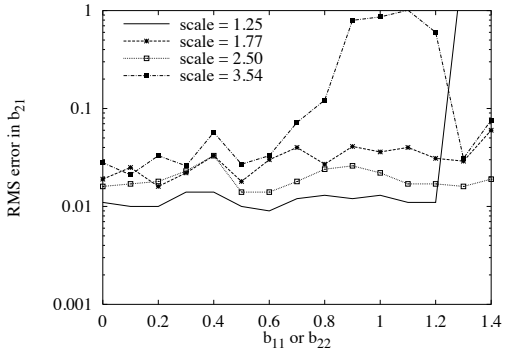
**Figure C.1.** RMS error in the affine parameters versus scale change ($b_{11}$) for the GauArea algorithm as a function of filter widths.



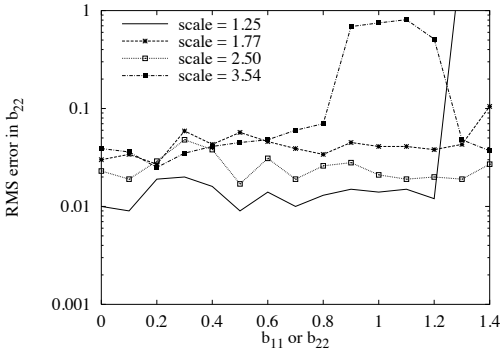

(a) RMS error in $b_{11}$



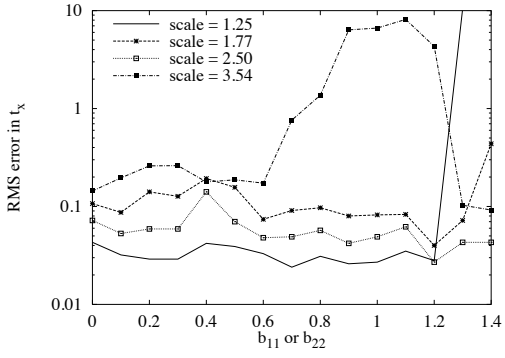

(b) RMS error in $b_{12}$



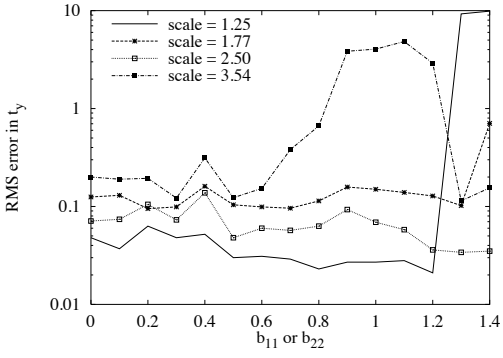

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

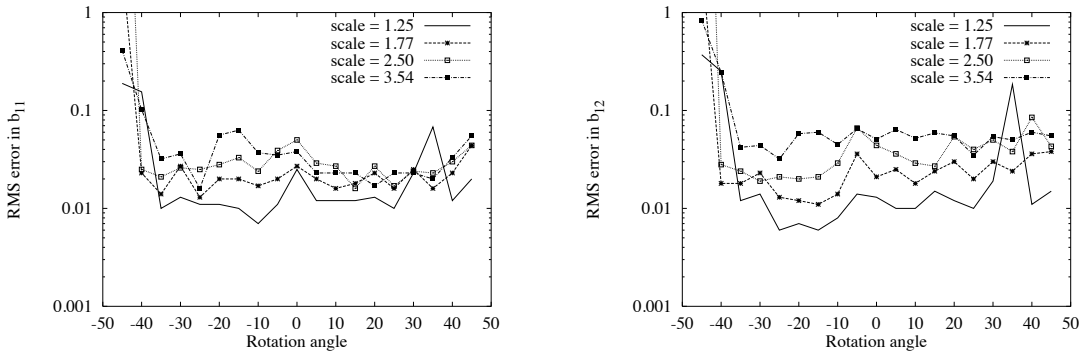



(e) RMS error in $t_x$




(f) RMS error in $t_y$

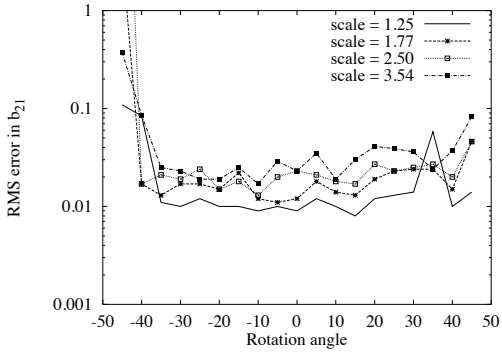
**Figure C.2.** RMS error in the affine parameters versus scale change ($b_{11}$) for the GauArea algorithm as a function of filter scale.



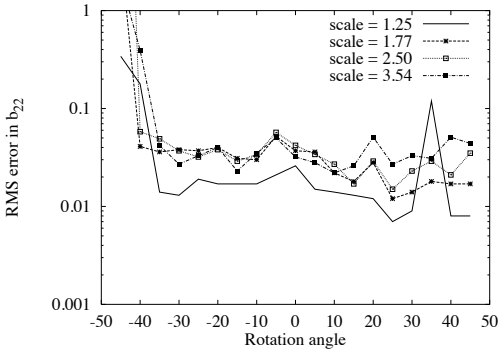

(a) RMS error in $b_{11}$



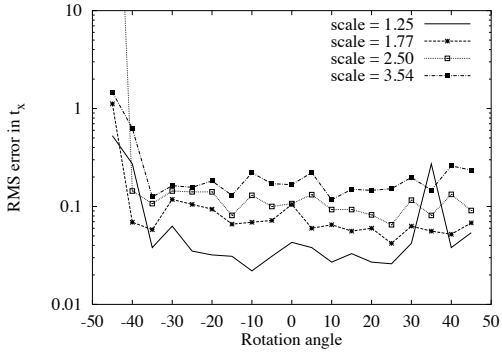

(b) RMS error in $b_{12}$



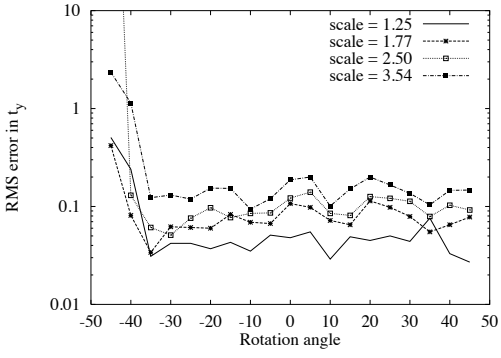

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

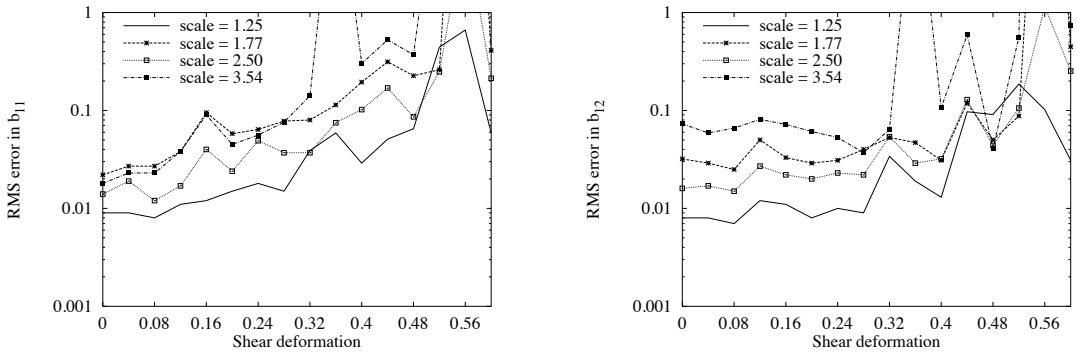



(e) RMS error in $t_x$




(f) RMS error in $t_y$

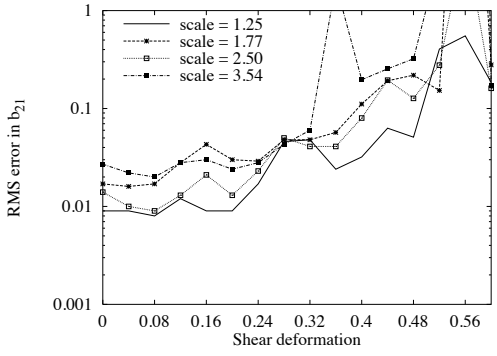
**Figure C.3.** RMS error in the affine parameters versus rotation angle for the GauArea algorithm as a function of filter scale.



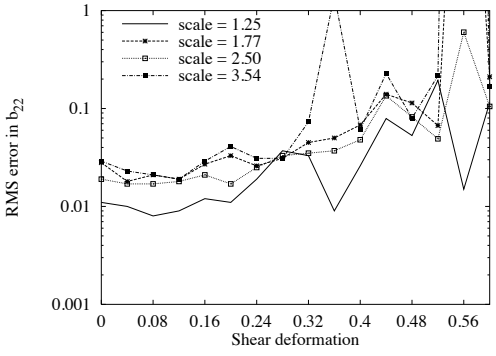

(a) RMS error in $b_{11}$



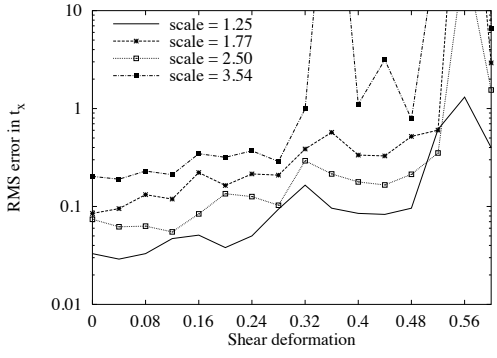

(b) RMS error in $b_{12}$



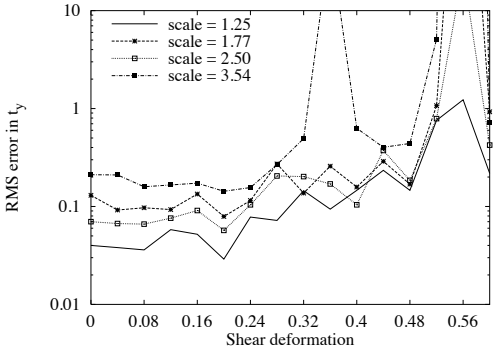

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

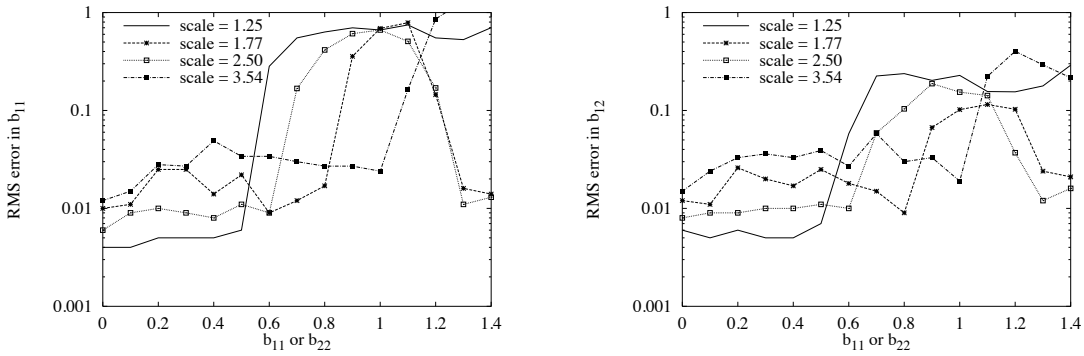



(e) RMS error in $t_x$




(f) RMS error in $t_y$

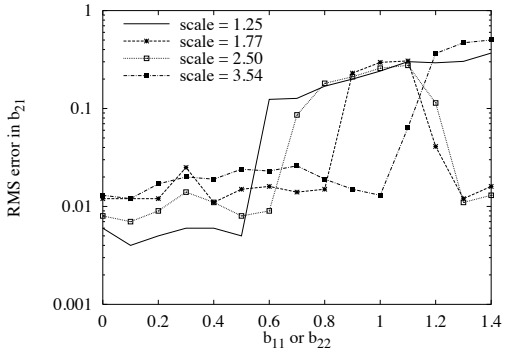
**Figure C.4.** RMS error in the affine parameters versus shear deformation for the GauArea algorithm as a function of filter scale.



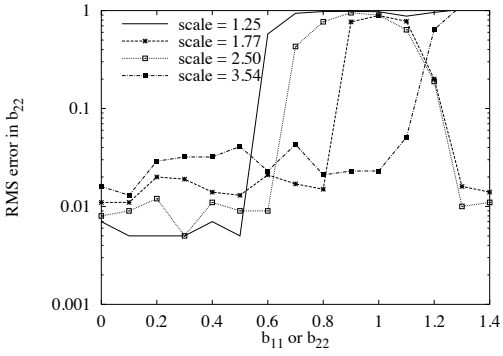

(a) RMS error in $b_{11}$



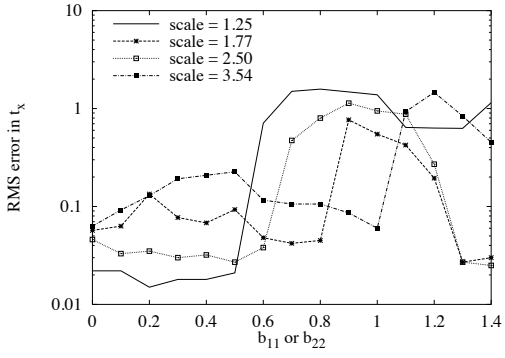

(b) RMS error in $b_{12}$



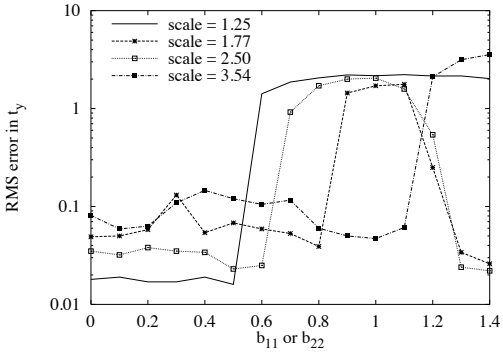

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

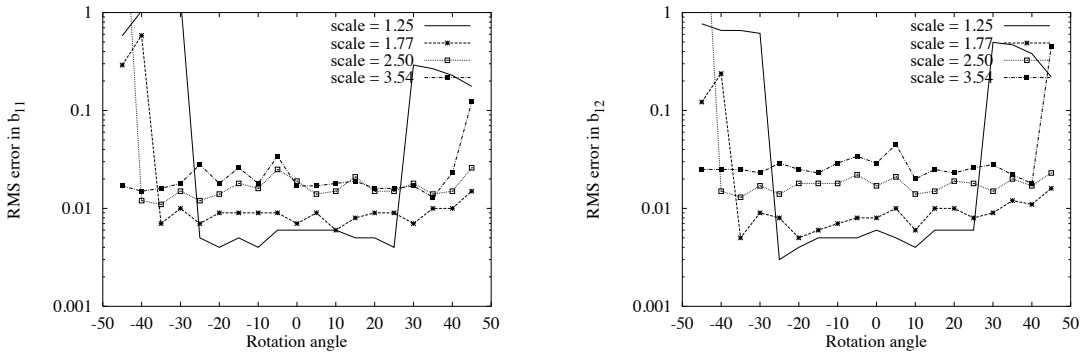



(e) RMS error in $t_x$




(f) RMS error in $t_y$

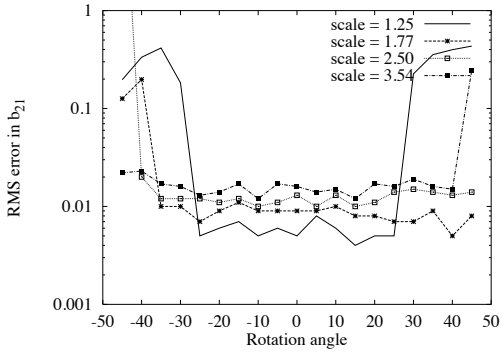
**Figure C.5.** RMS error in the affine parameters versus scale change ($b_{11}$) for the DGauArea algorithm as a function of filter scale.



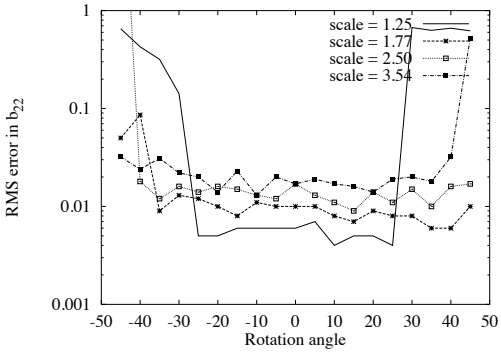

(a) RMS error in $b_{11}$



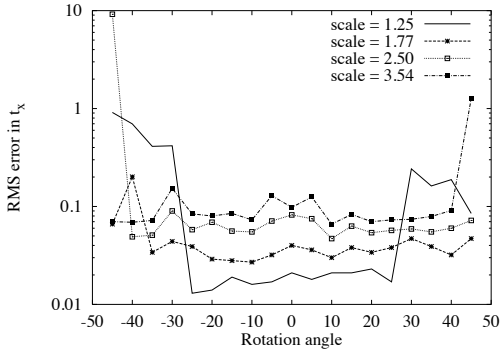

(b) RMS error in $b_{12}$



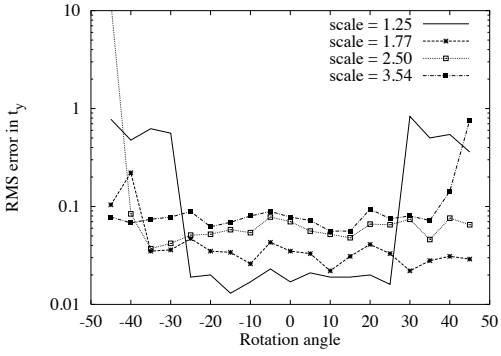

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

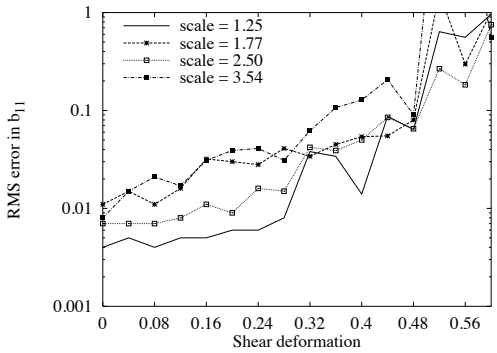



(e) RMS error in $t_x$

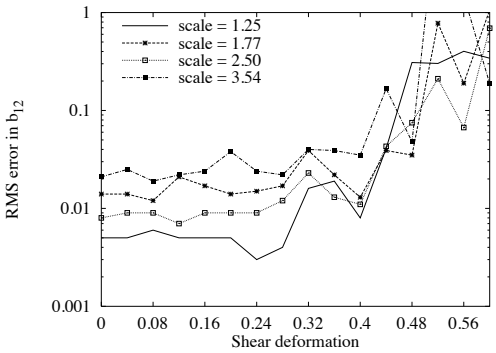



(f) RMS error in $t_y$

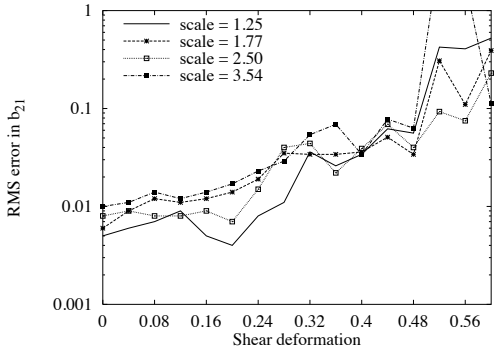
**Figure C.6.** RMS error in the affine parameters versus rotation angle for the DGauArea algorithm as a function of filter scale.



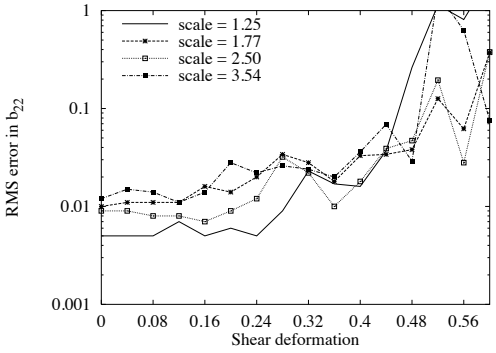

(a) RMS error in $b_{11}$



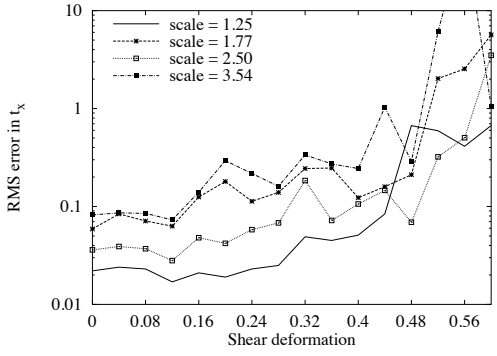

(b) RMS error in $b_{12}$



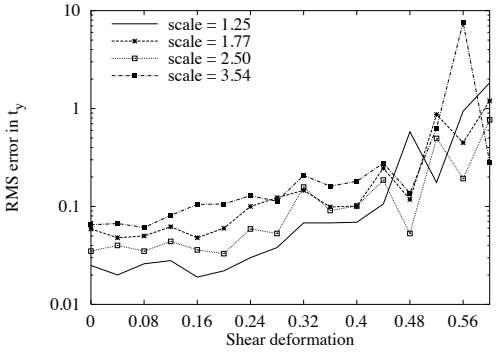

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

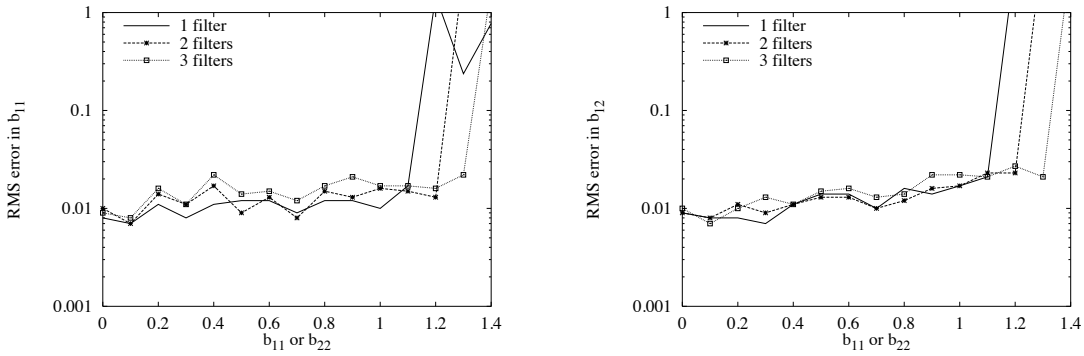



(e) RMS error in $t_x$




(f) RMS error in $t_y$

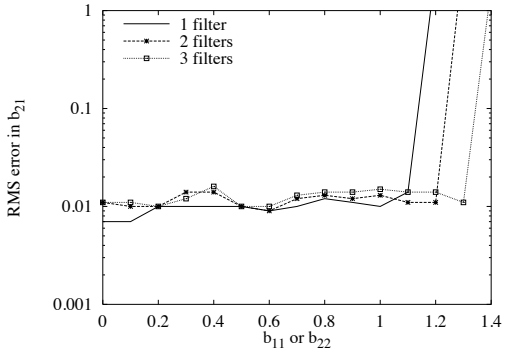
**Figure C.7.** RMS error in the affine parameters versus shear deformation for the DGauArea algorithm as a function of filter scale.



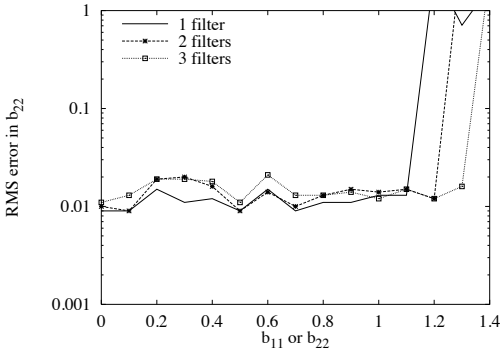

(a) RMS error in $b_{11}$



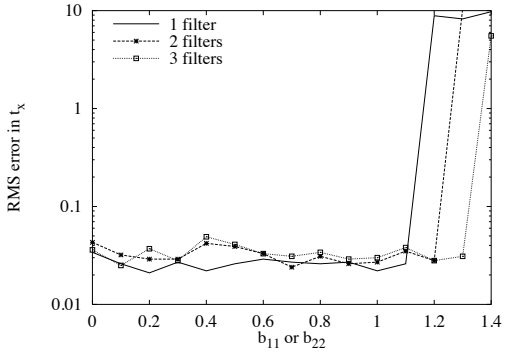

(b) RMS error in $b_{12}$



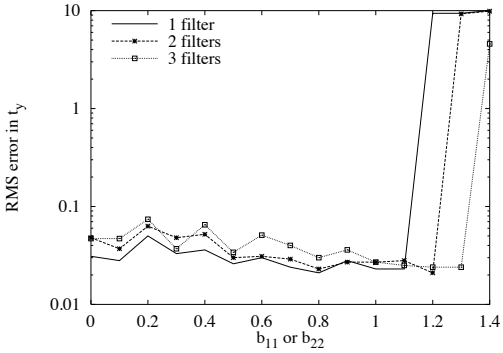

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

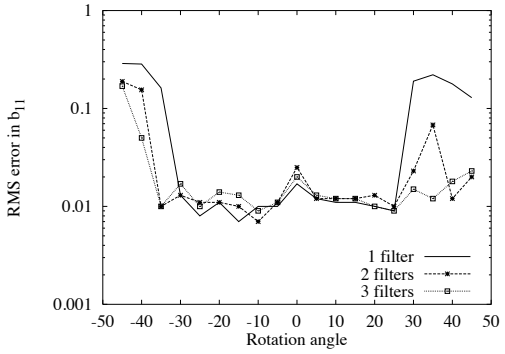



(e) RMS error in $t_x$

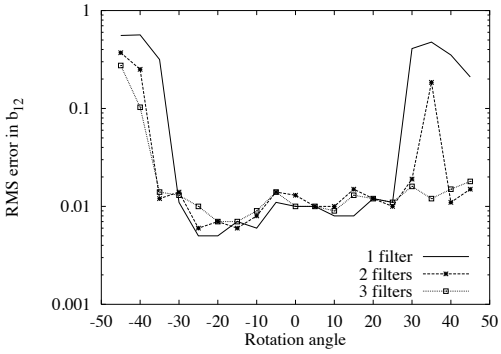



(f) RMS error in $t_y$

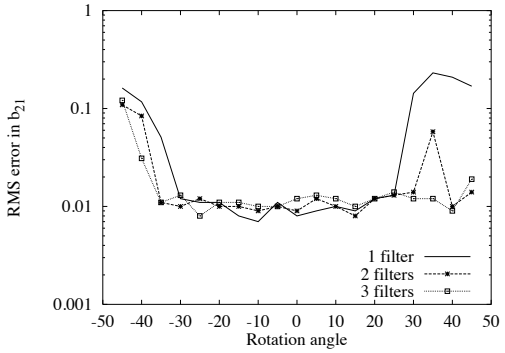
**Figure C.8.** RMS error in the affine parameters versus scale change ($b_{11}$) for the GauArea algorithm as a function of the number of filters used.



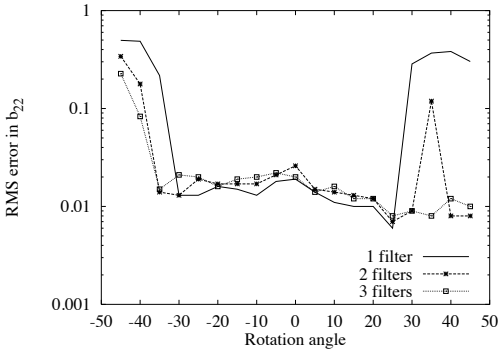

(a) RMS error in $b_{11}$



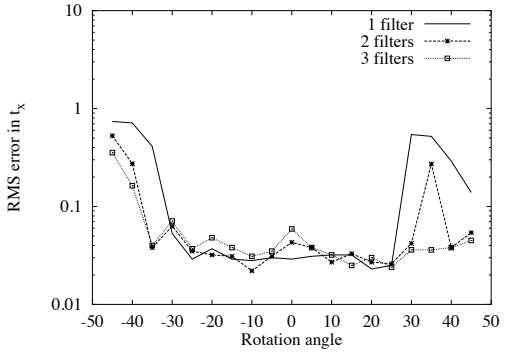

(b) RMS error in $b_{12}$



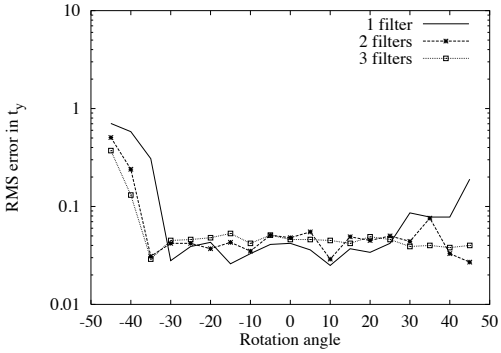

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

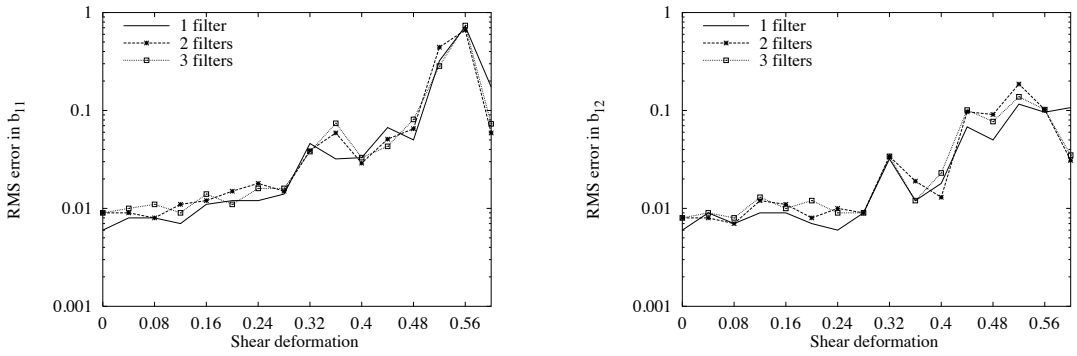



(e) RMS error in $t_x$



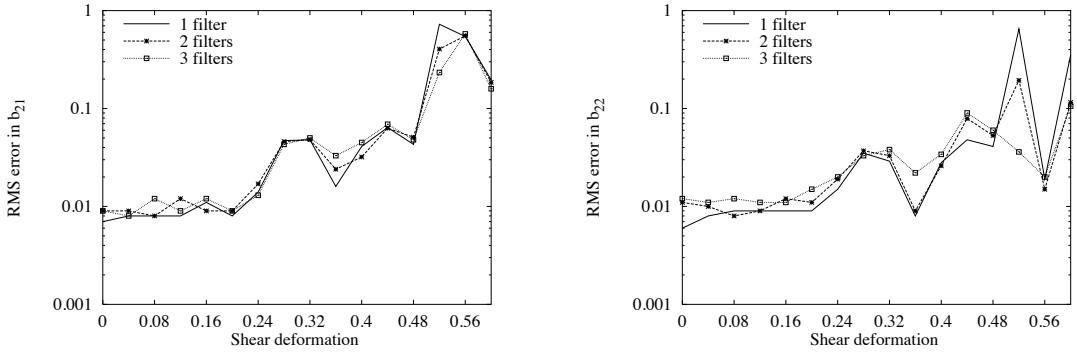

(f) RMS error in $t_y$

**Figure C.9.** RMS error in the affine parameters versus rotation angle for the GauArea algorithm as a function of the number of filters used.



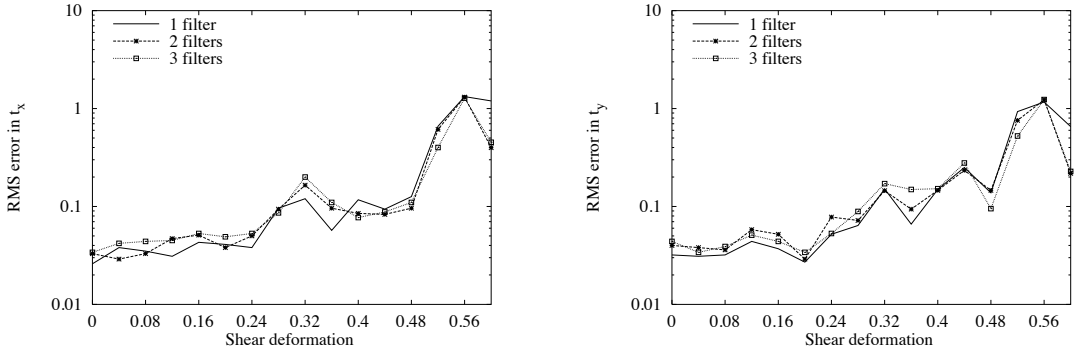

(a) RMS error in $b_{11}$

(b) RMS error in $b_{12}$




(c) RMS error in $b_{21}$

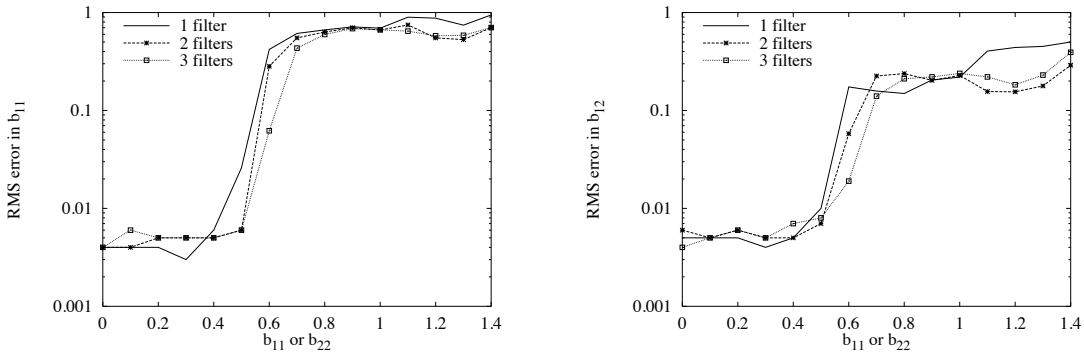
(d) RMS error in $b_{22}$




(e) RMS error in $t_x$

(f) RMS error in $t_y$

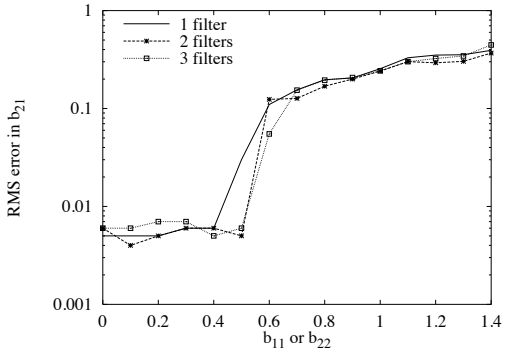
**Figure C.10.** RMS error in the affine parameters versus shear deformation for the GauArea algorithm as a function of the number of filters used.



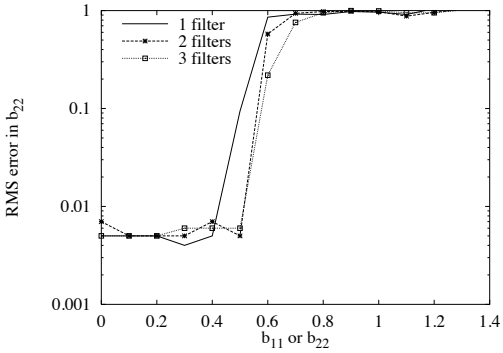

(a) RMS error in $b_{11}$



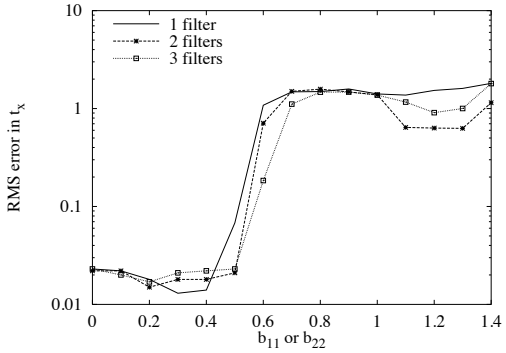

(b) RMS error in $b_{12}$



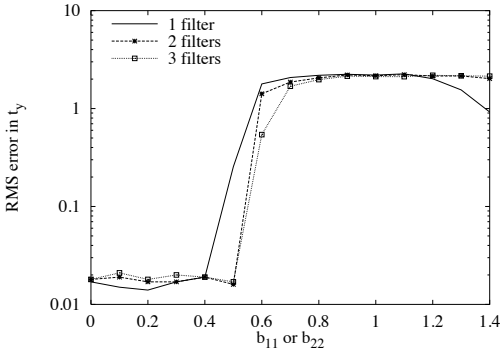

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

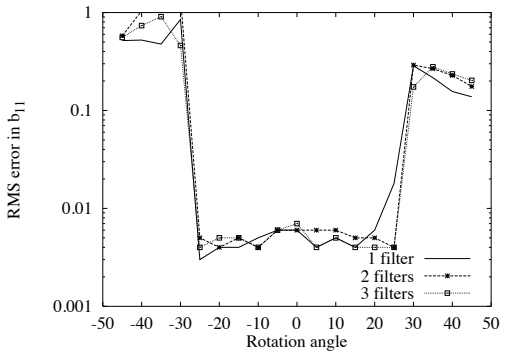



(e) RMS error in $t_x$

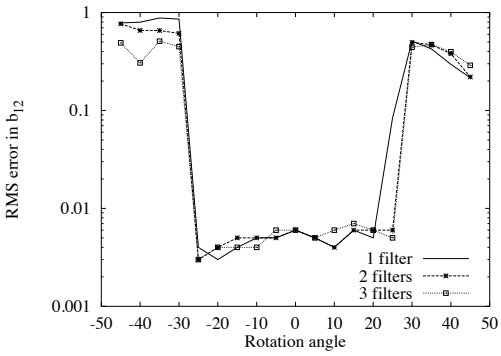



(f) RMS error in $t_y$

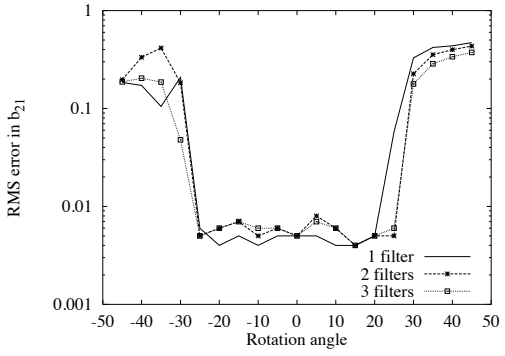
**Figure C.11.** RMS error in the affine parameters versus scale change ($b_{11}$) for the DGauArea algorithm as a function of the number of filters used.



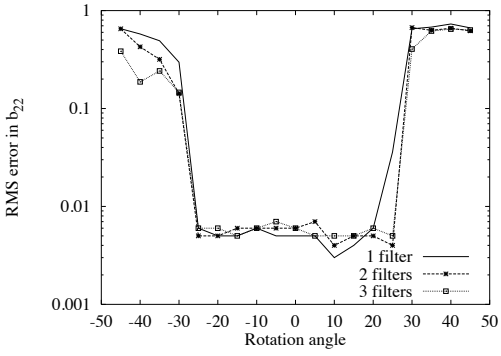

(a) RMS error in $b_{11}$



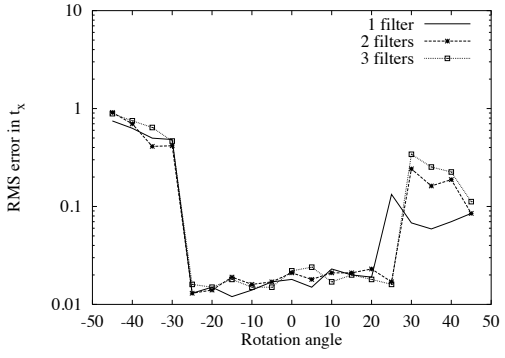

(b) RMS error in $b_{12}$



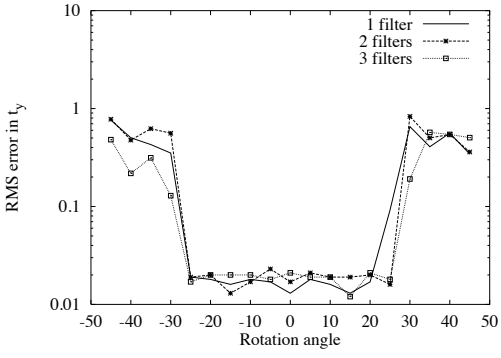

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

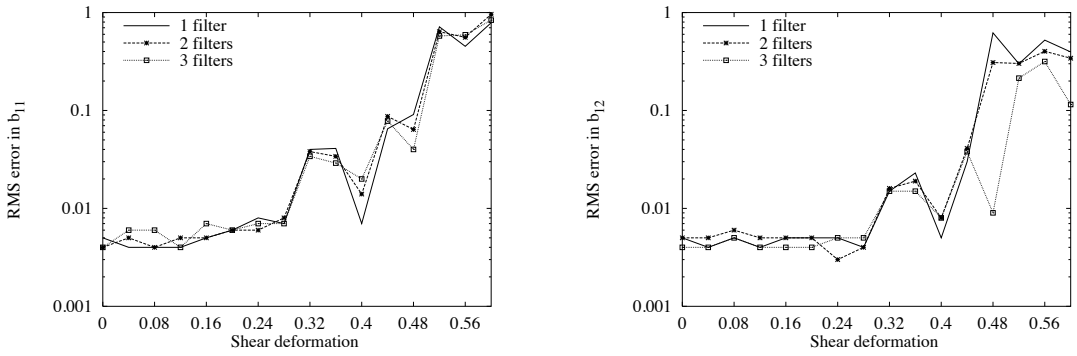



(e) RMS error in $t_x$




(f) RMS error in $t_y$

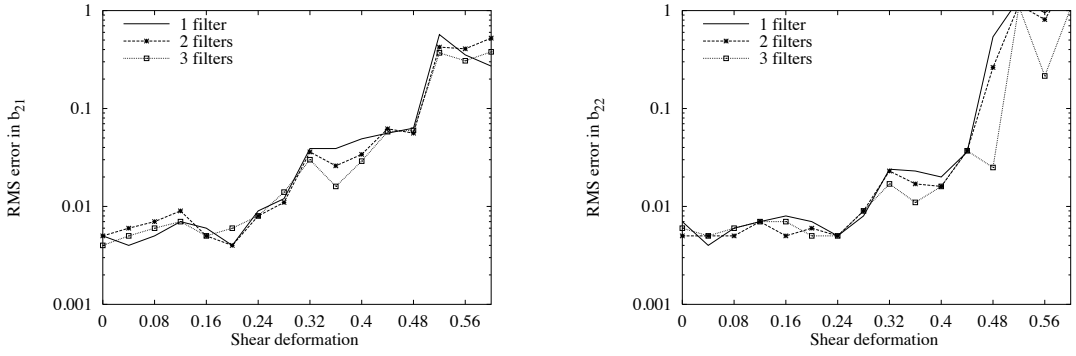
**Figure C.12.** RMS error in the affine parameters versus rotation angle for the DGauArea algorithm as a function of the number of filters used.




(a) RMS error in $b_{11}$



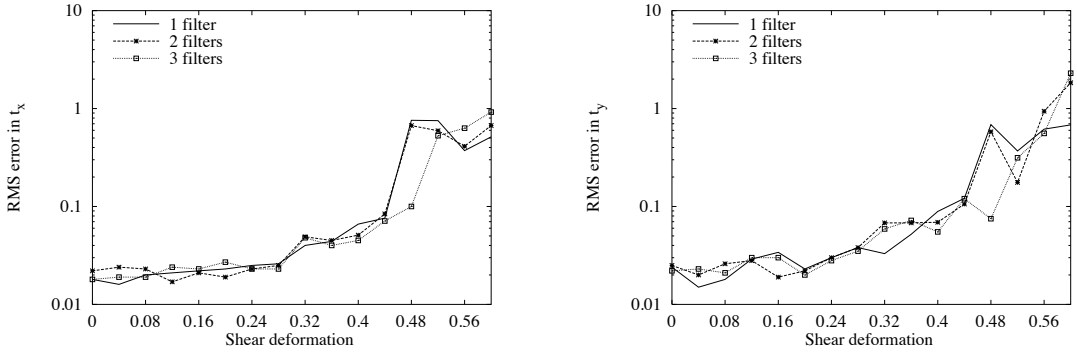

(b) RMS error in $b_{12}$




(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

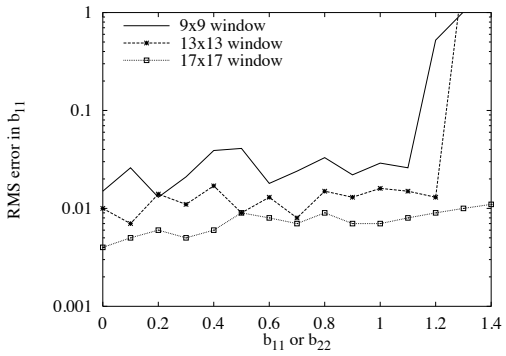



(e) RMS error in $t_x$

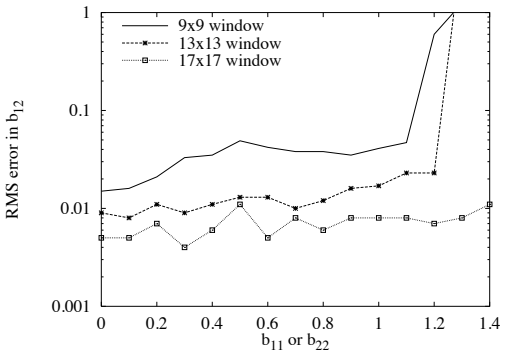



(f) RMS error in $t_y$

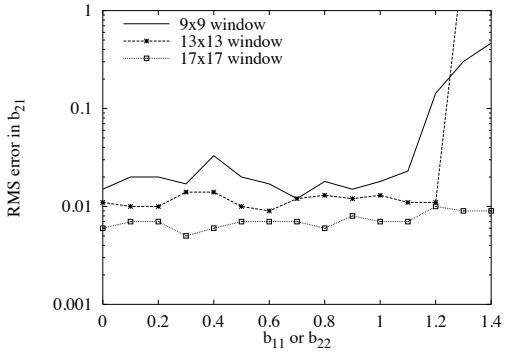
**Figure C.13.** RMS error in the affine parameters versus shear deformation for the DGauArea algorithm as a function of the number of filters used.



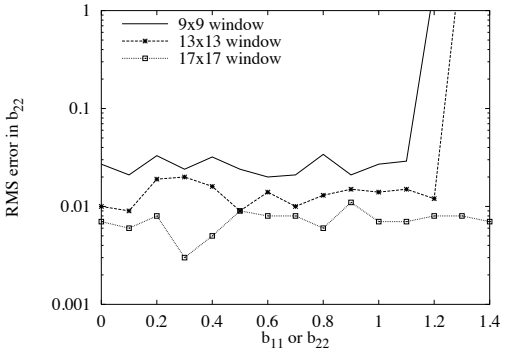

(a) RMS error in $b_{11}$



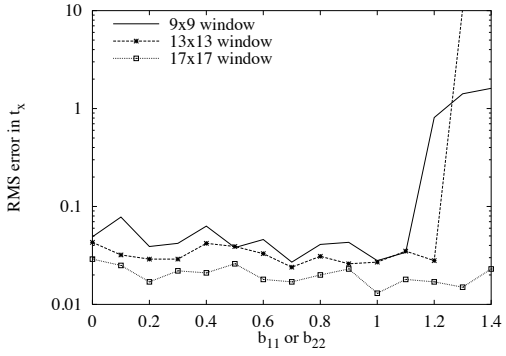

(b) RMS error in $b_{12}$



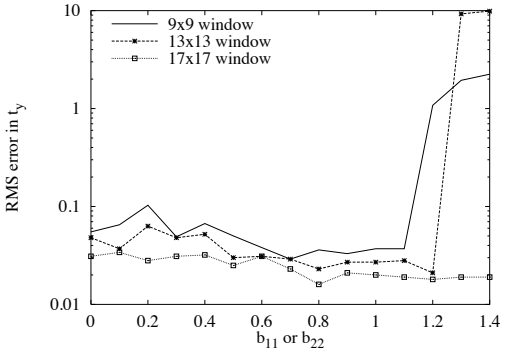

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

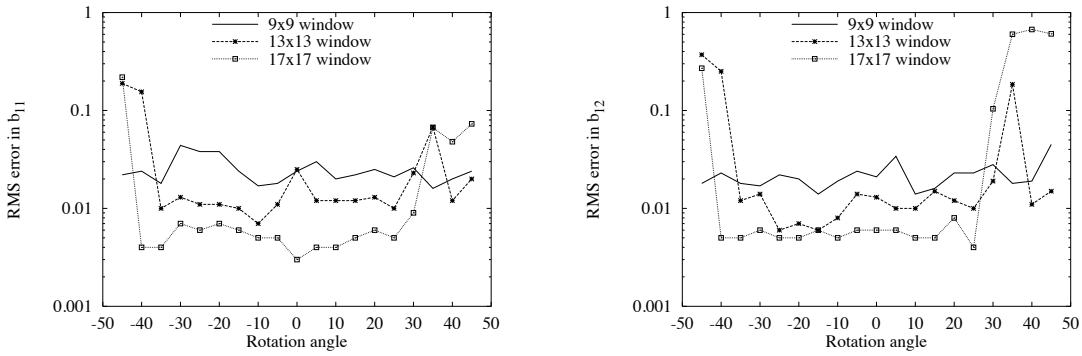



(e) RMS error in $t_x$




(f) RMS error in $t_y$

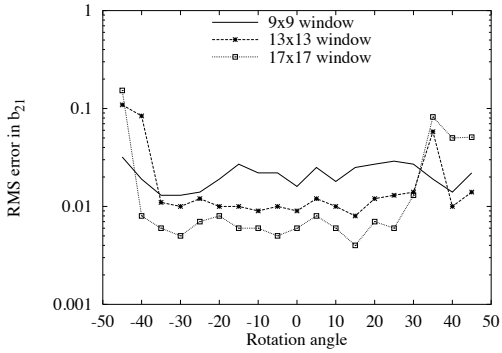
**Figure C.14.** RMS error in the affine parameters versus scale change ($b_{11}$) for the GauArea algorithm as a function of windowsize.



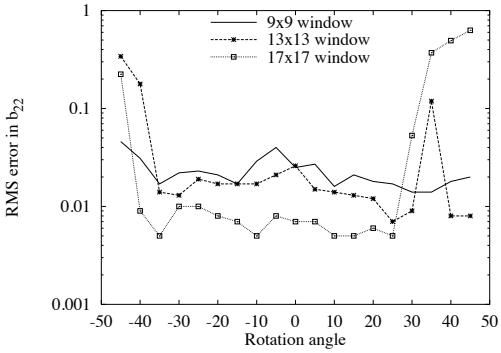

(a) RMS error in $b_{11}$



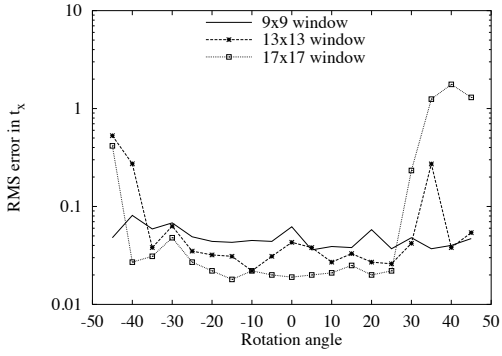

(b) RMS error in $b_{12}$



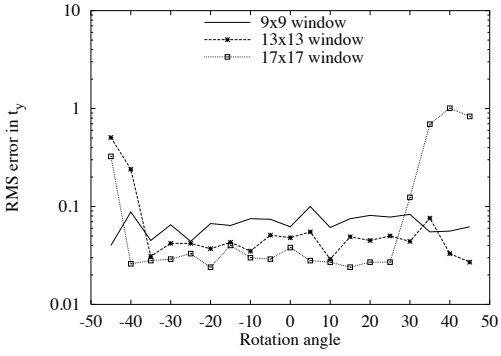

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

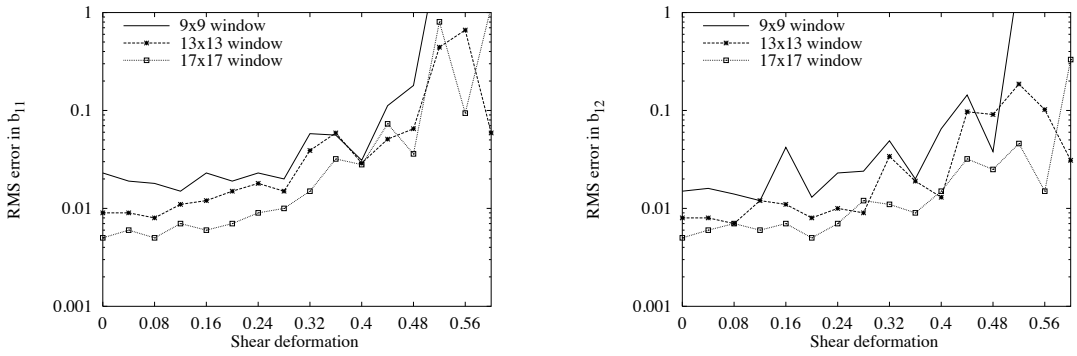



(e) RMS error in $t_x$




(f) RMS error in $t_y$

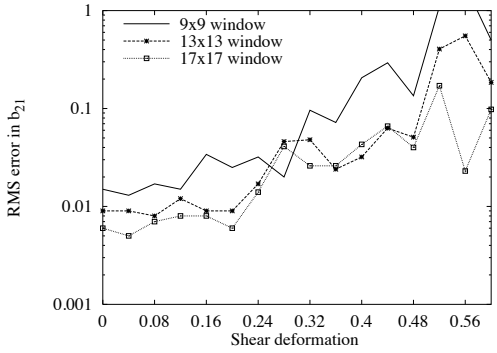
**Figure C.15.** RMS error in the affine parameters versus rotation angle for the GauArea algorithm as a function of windowsize.



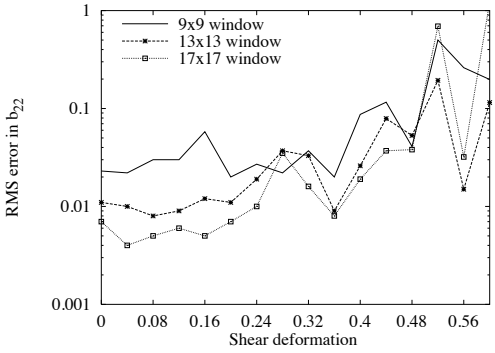

(a) RMS error in $b_{11}$



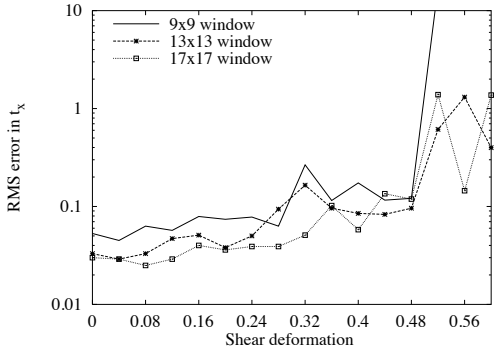

(b) RMS error in $b_{12}$



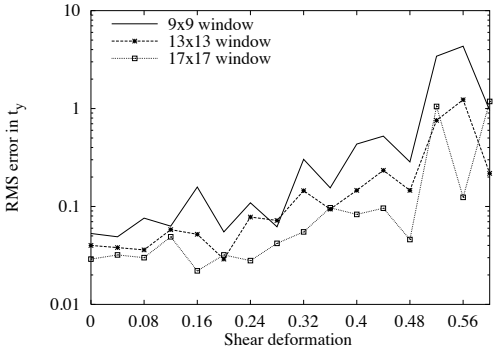

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

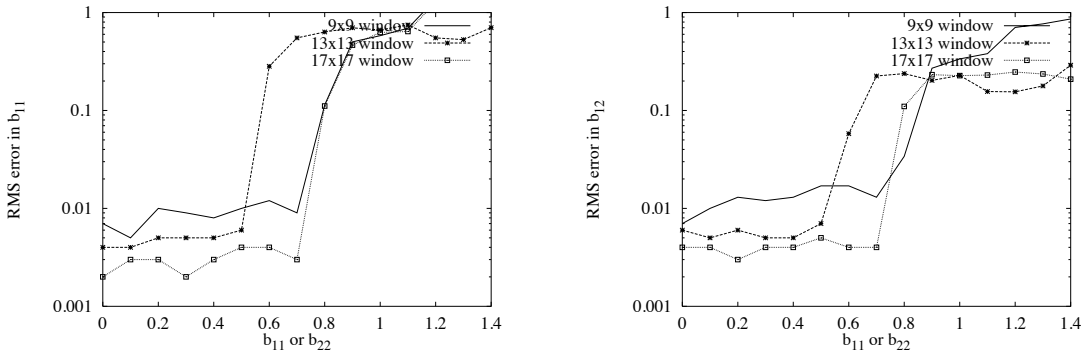



(e) RMS error in $t_x$




(f) RMS error in $t_y$

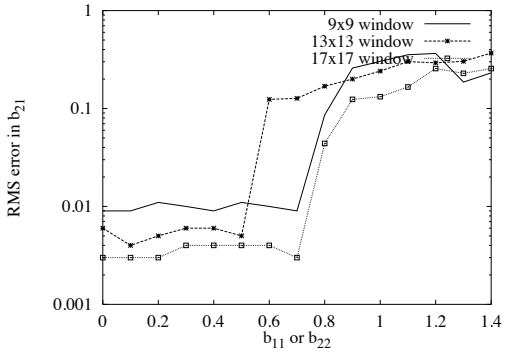
**Figure C.16.** RMS error in the affine parameters versus shear deformation for the GauArea algorithm as a function of windowsize.



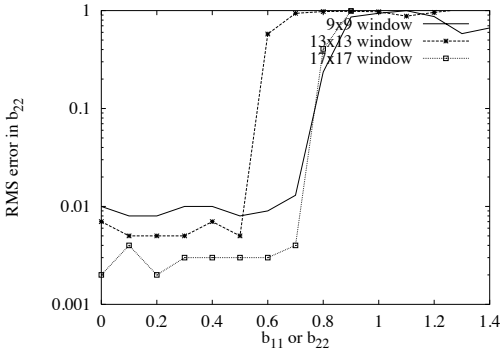

(a) RMS error in $b_{11}$



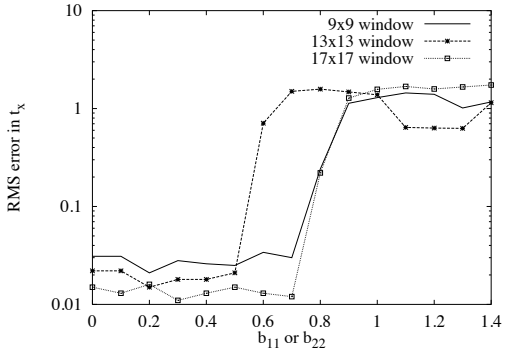

(b) RMS error in $b_{12}$



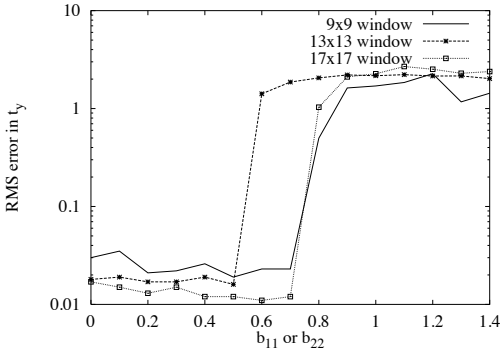

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

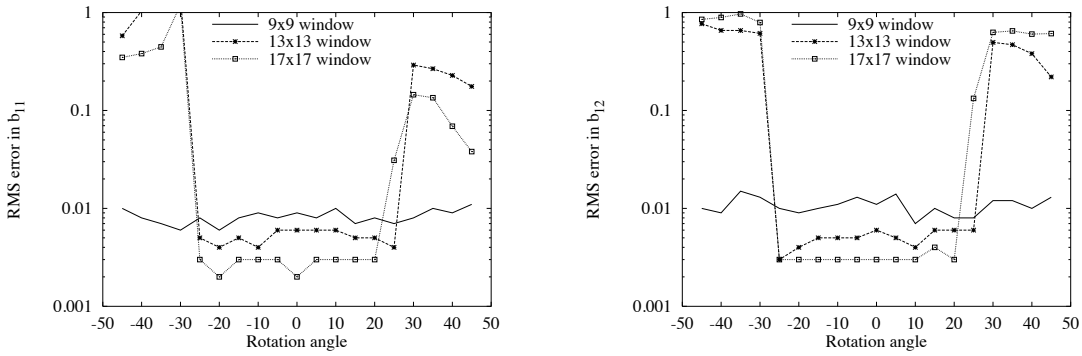



(e) RMS error in $t_x$




(f) RMS error in $t_y$

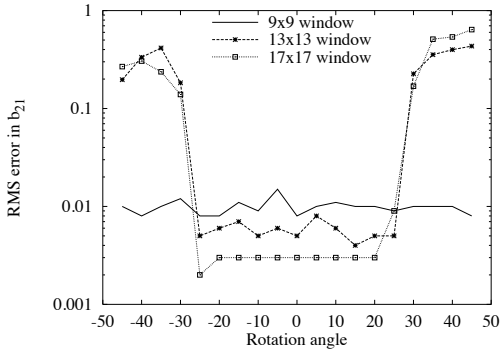
**Figure C.17.** RMS error in the affine parameters versus scale change ($b_{11}$) for the DGauArea algorithm as a function of windowsize.



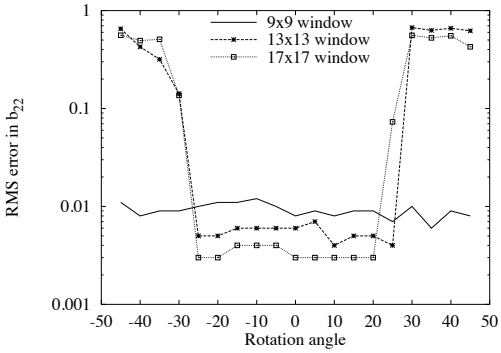

(a) RMS error in $b_{11}$



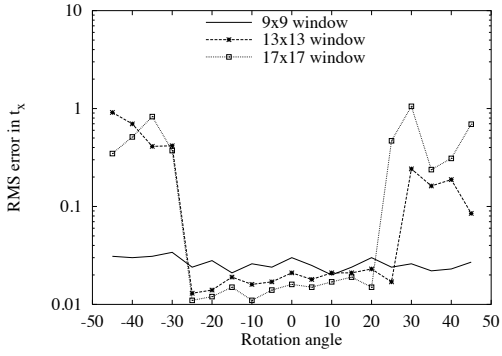

(b) RMS error in $b_{12}$



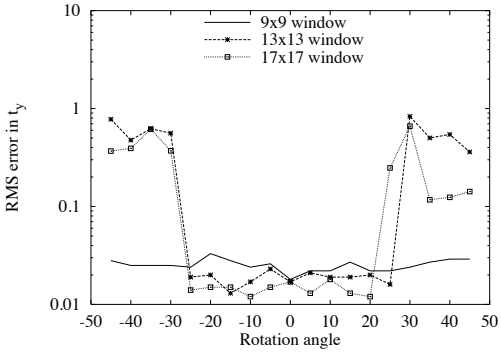

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

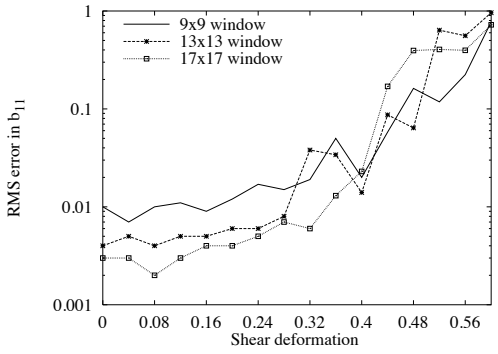



(e) RMS error in $t_x$

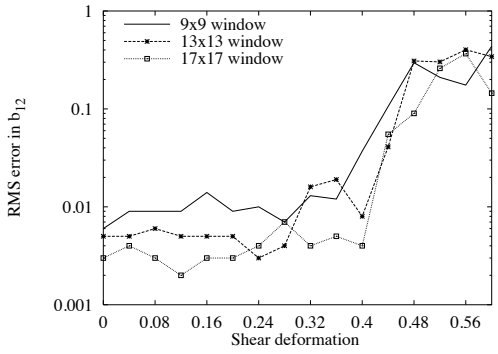



(f) RMS error in $t_y$

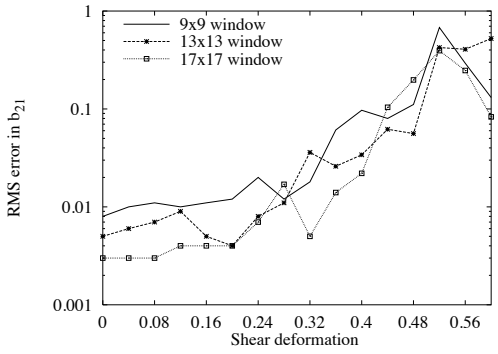
**Figure C.18.** RMS error in the affine parameters versus rotation angle for the DGauArea algorithm as a function of windowsize.



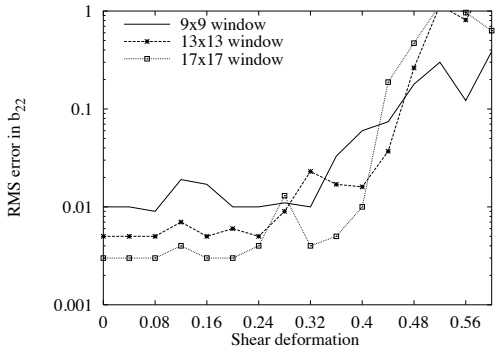

(a) RMS error in $b_{11}$



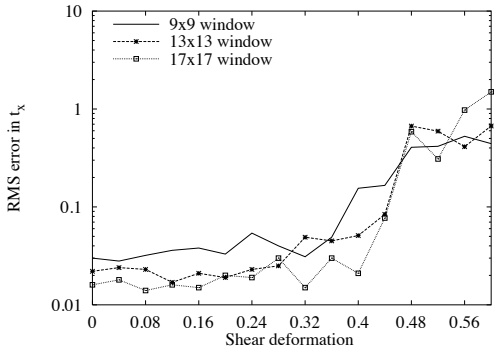

(b) RMS error in $b_{12}$



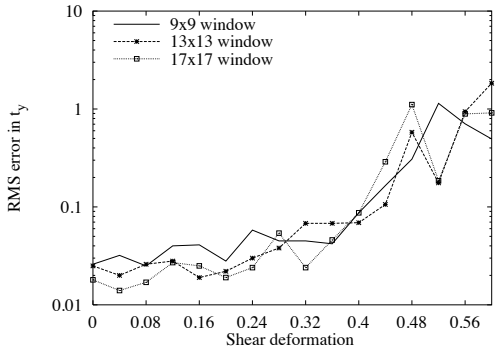

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

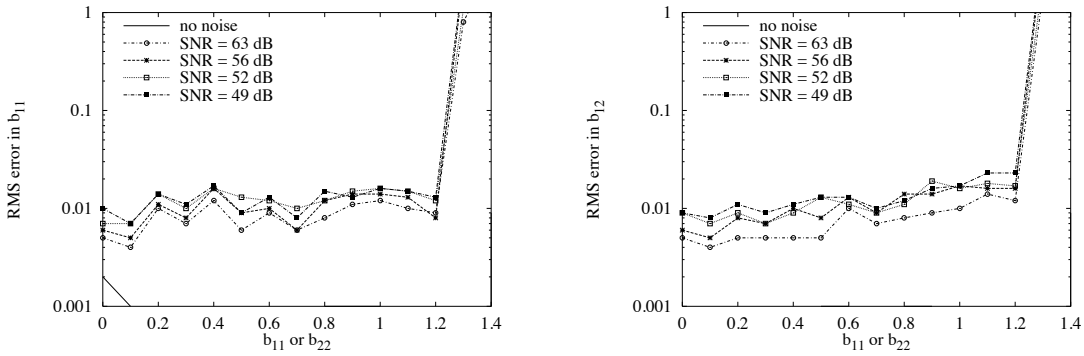



(e) RMS error in $t_x$




(f) RMS error in $t_y$

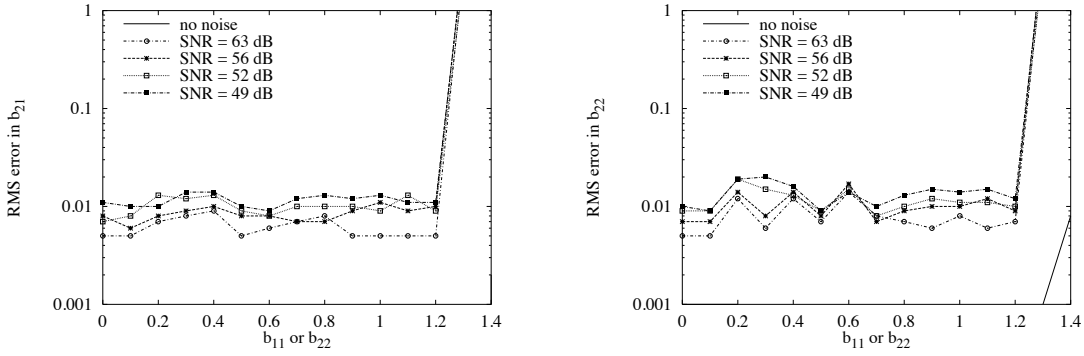
**Figure C.19.** RMS error in the affine parameters versus shear deformation for the DGauArea algorithm as a function of windowsize.




(a) RMS error in $b_{11}$



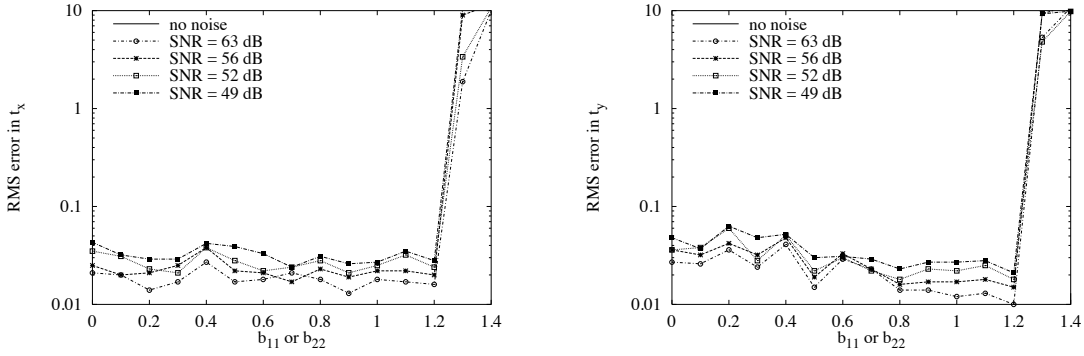

(b) RMS error in $b_{12}$




(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

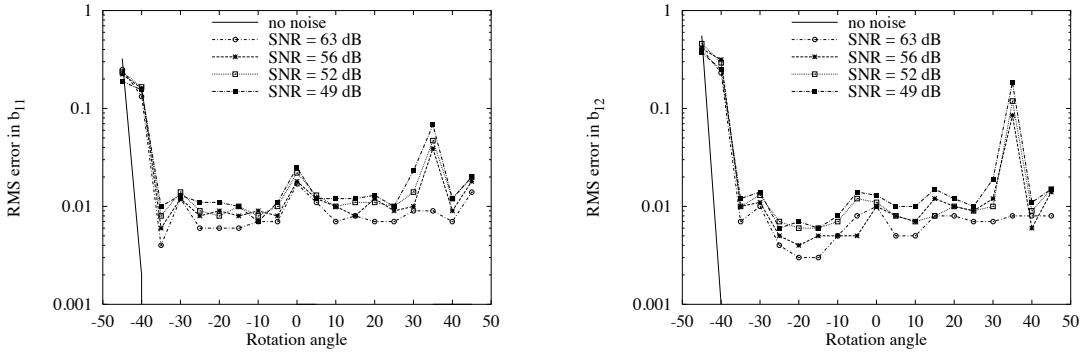



(e) RMS error in $t_x$



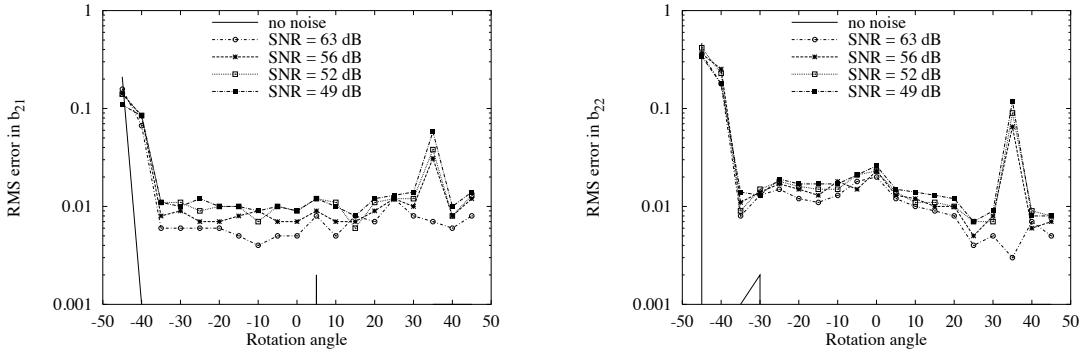

(f) RMS error in $t_y$

**Figure C.20.** RMS error in the affine parameters versus scale change ($b_{11}$) for the GauArea algorithm as a function of the noise.



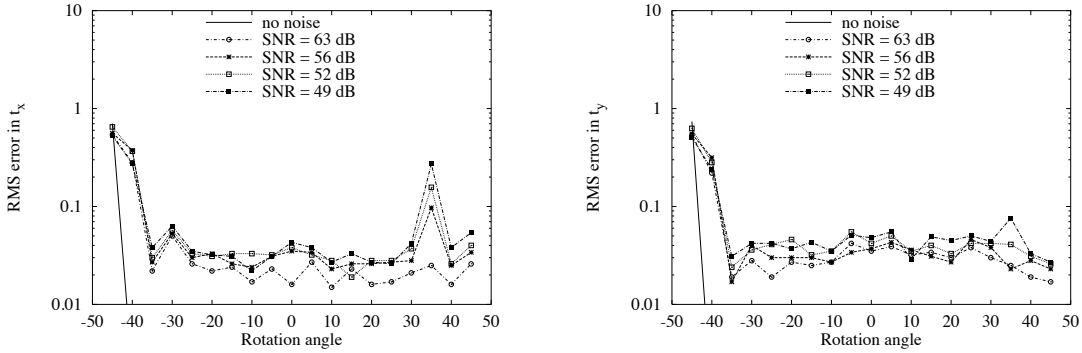

(a) RMS error in $b_{11}$

(b) RMS error in $b_{12}$




(c) RMS error in $b_{21}$

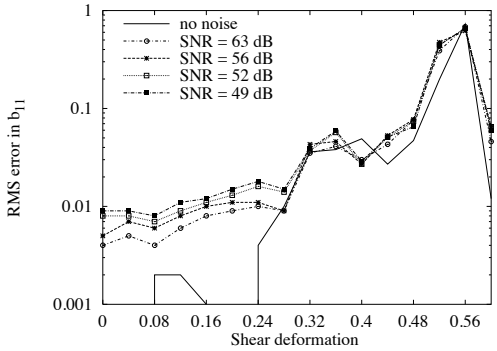
(d) RMS error in $b_{22}$



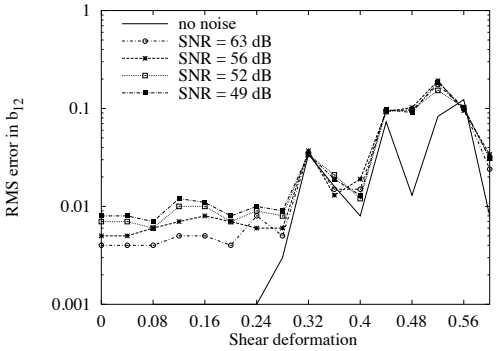

(e) RMS error in $t_x$

(f) RMS error in $t_y$

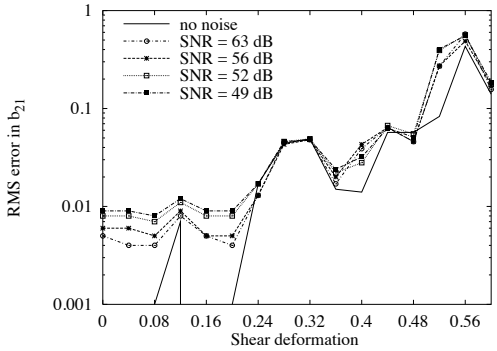
**Figure C.21.** RMS error in the affine parameters versus rotation angle for the GauArea algorithm as a function of the noise.



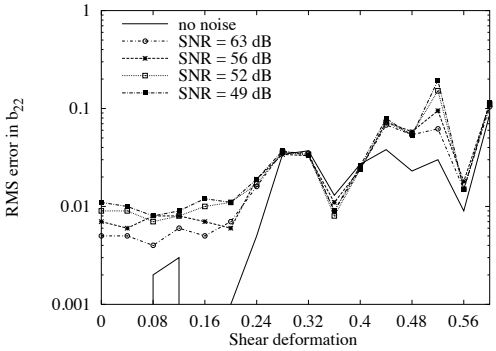

(a) RMS error in $b_{11}$



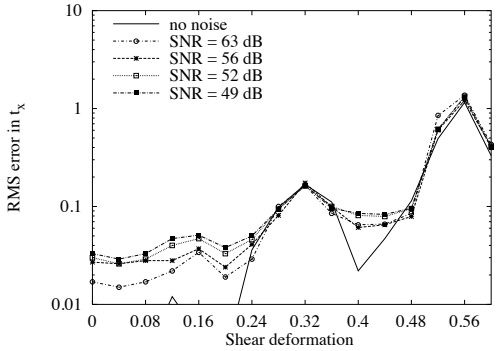

(b) RMS error in $b_{12}$



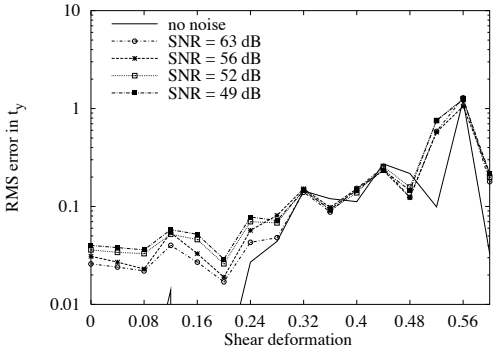

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

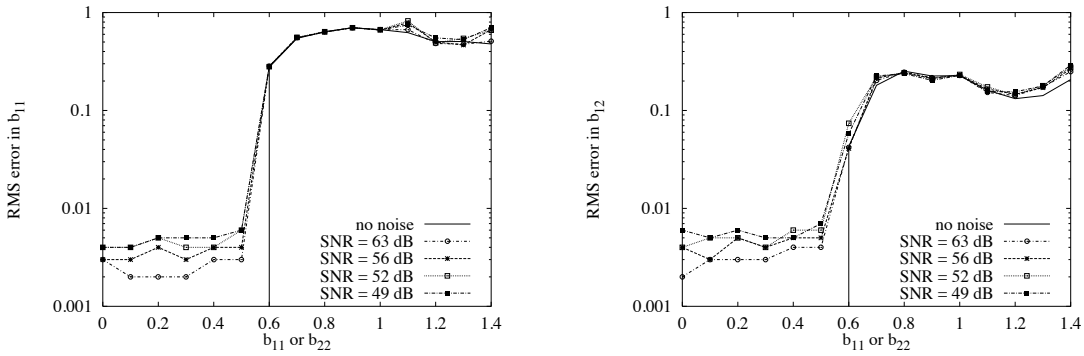



(e) RMS error in $t_x$




(f) RMS error in $t_y$

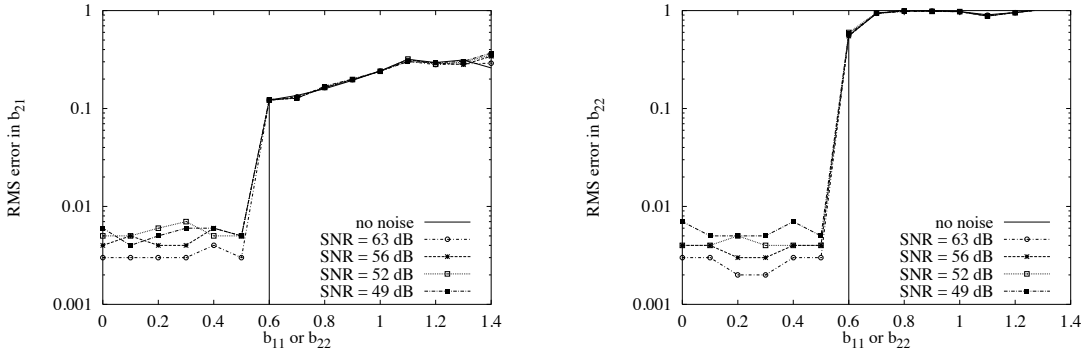
**Figure C.22.** RMS error in the affine parameters versus shear deformation for the GauArea algorithm as a function of the noise.




(a) RMS error in $b_{11}$



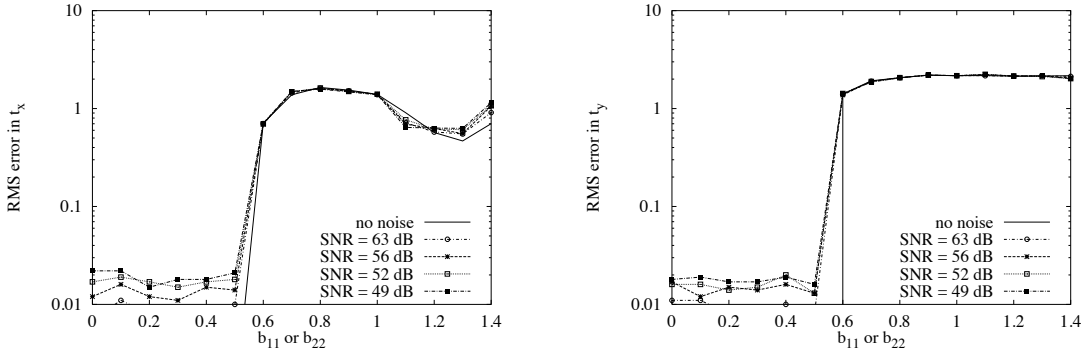

(b) RMS error in $b_{12}$




(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

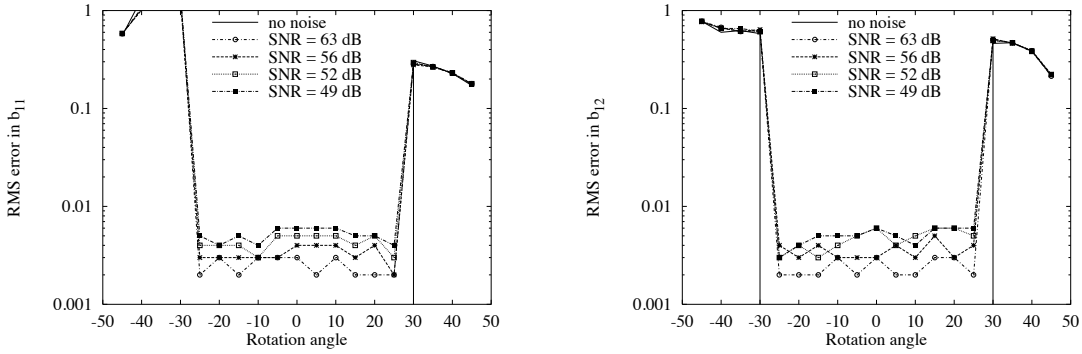



(e) RMS error in $t_x$



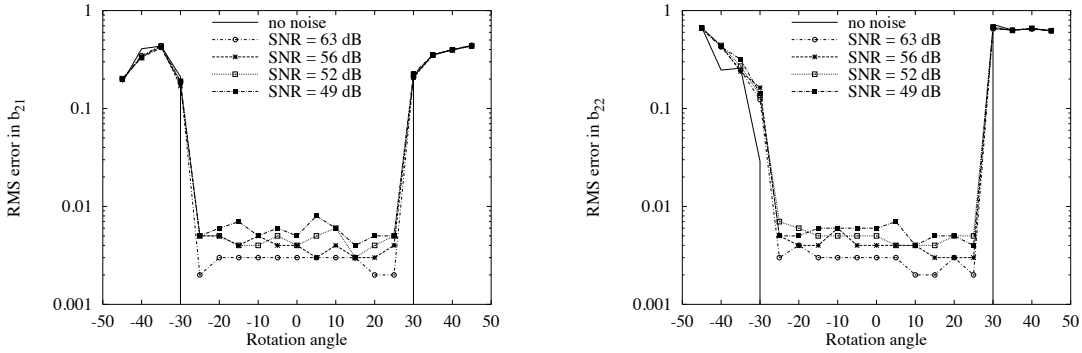

(f) RMS error in $t_y$

**Figure C.23.** RMS error in the affine parameters versus scale change ($b_{11}$) for the DGauArea algorithm as a function of the noise.



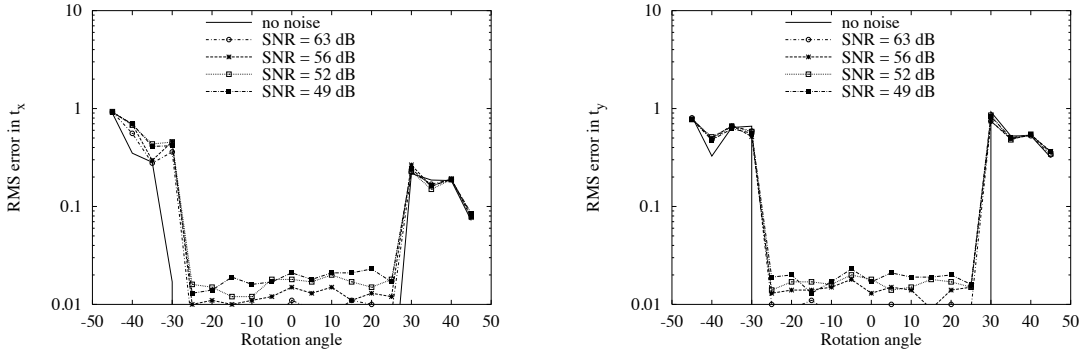

(a) RMS error in $b_{11}$

(b) RMS error in $b_{12}$




(c) RMS error in $b_{21}$

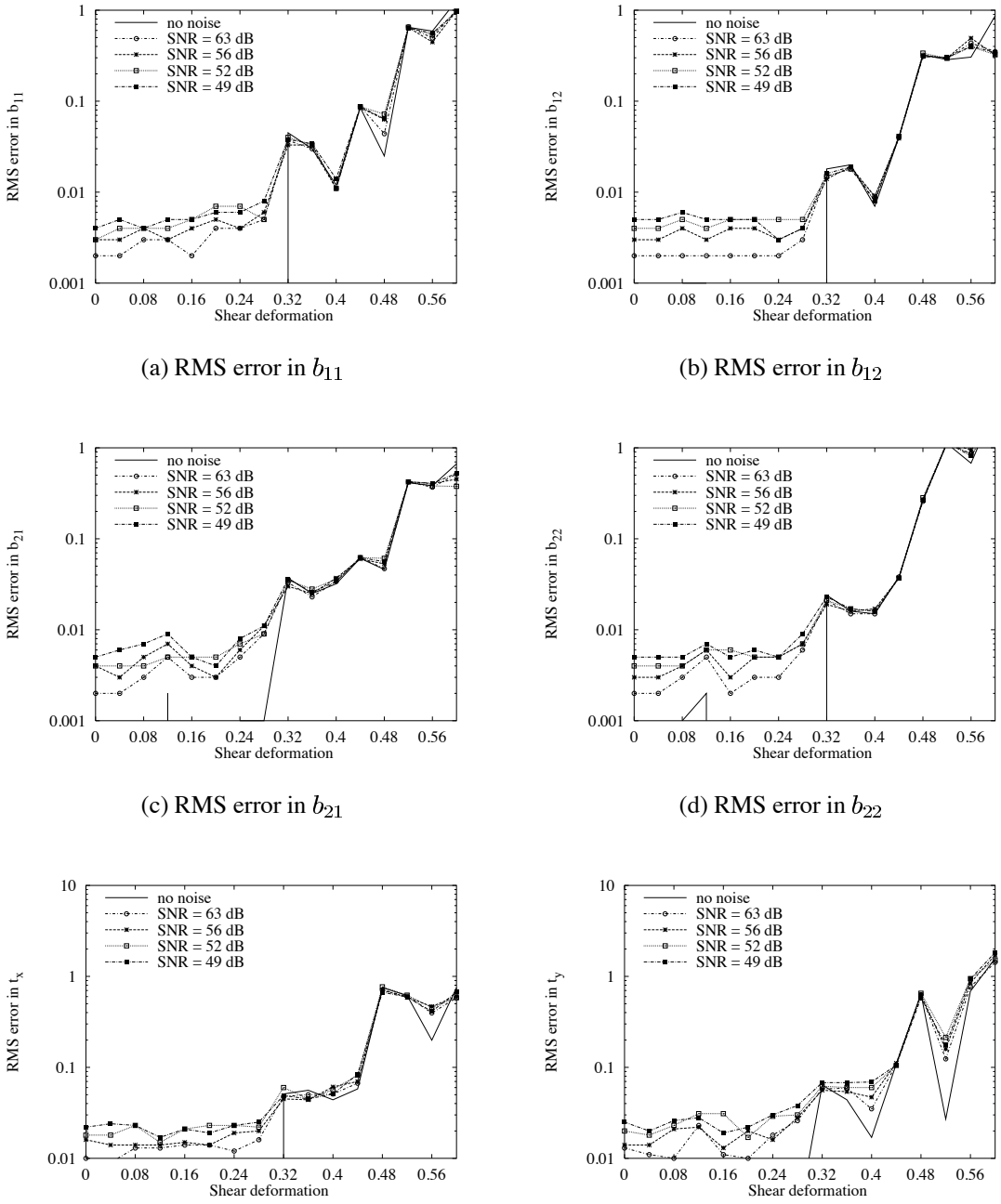
(d) RMS error in $b_{22}$




(e) RMS error in $t_x$

(f) RMS error in $t_y$

**Figure C.24.** RMS error in the affine parameters versus rotation angle for the DGauArea algorithm as a function of the noise.




(a) RMS error in $b_{11}$

(b) RMS error in $b_{12}$

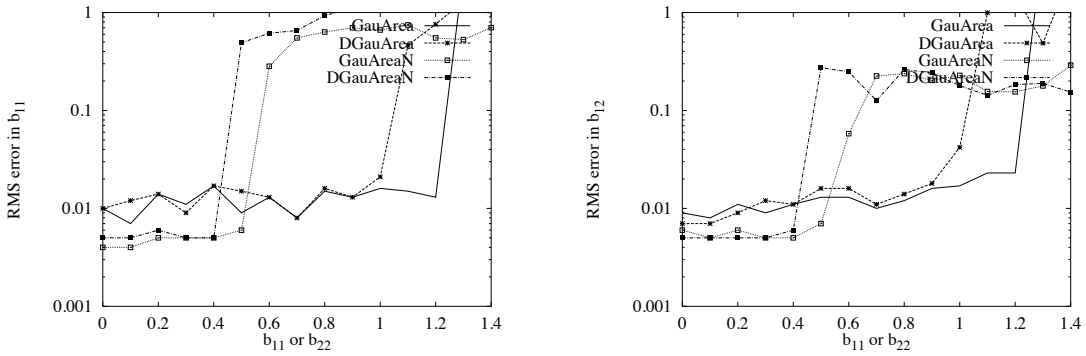
(c) RMS error in $b_{21}$

(d) RMS error in $b_{22}$

(e) RMS error in $t_x$

(f) RMS error in $t_y$

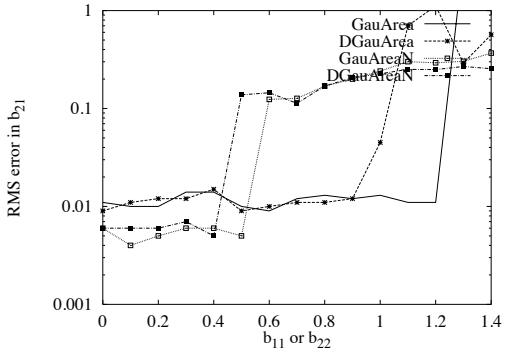
**Figure C.25.** RMS error in the affine parameters versus shear deformation for the DGauArea algorithm as a function of the noise.



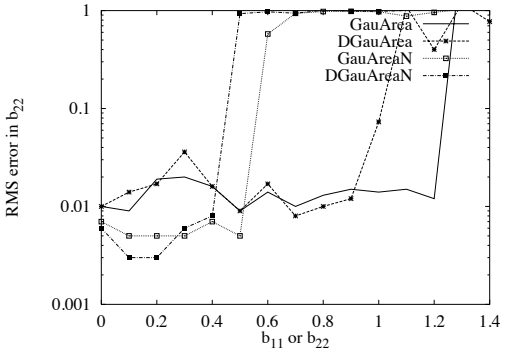

(a) RMS error in $b_{11}$



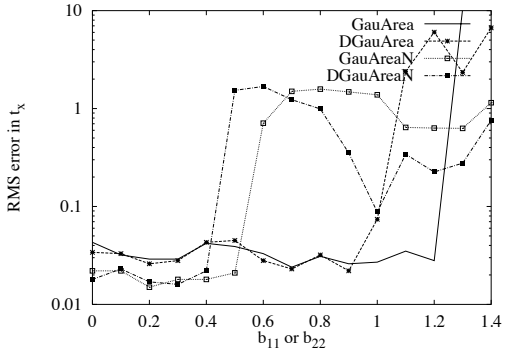

(b) RMS error in $b_{12}$



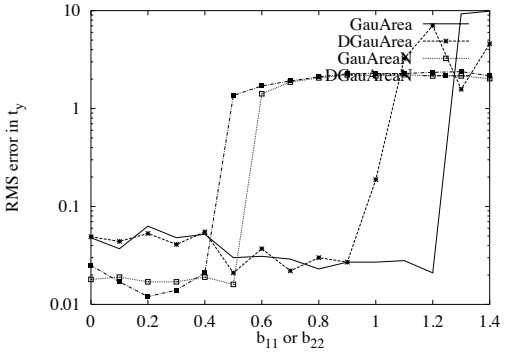

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

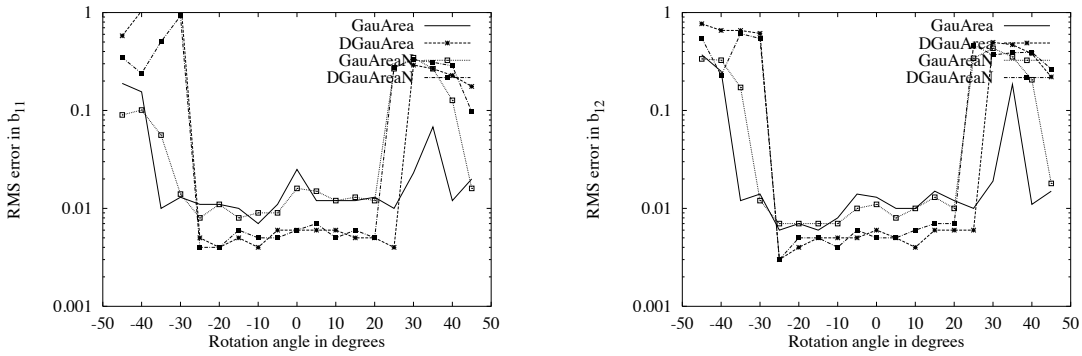



(e) RMS error in $t_x$




(f) RMS error in $t_y$

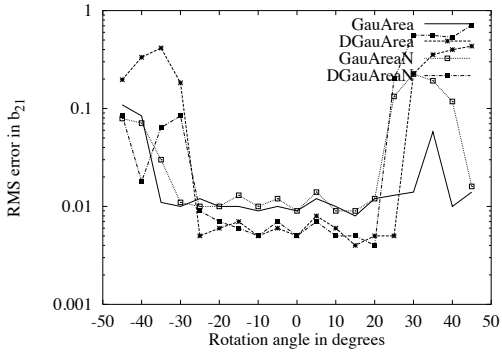
**Figure C.26.** Comparison of RMS errors for four different algorithms when the image undergoes a scale change. The error in the affine parameters is plotted against ($b_{11}$).



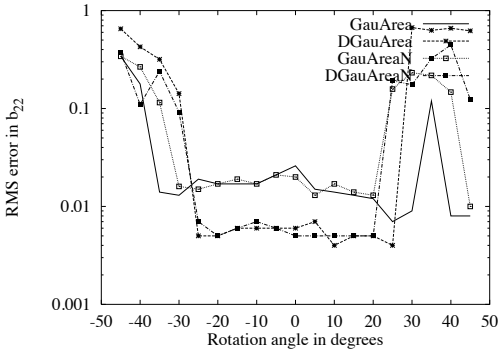

(a) RMS error in $b_{11}$



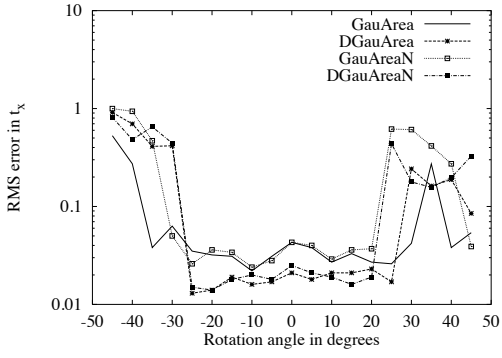

(b) RMS error in $b_{12}$



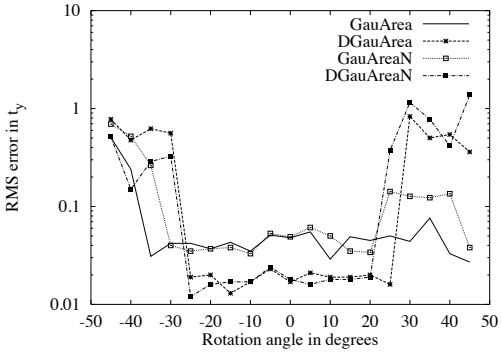

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

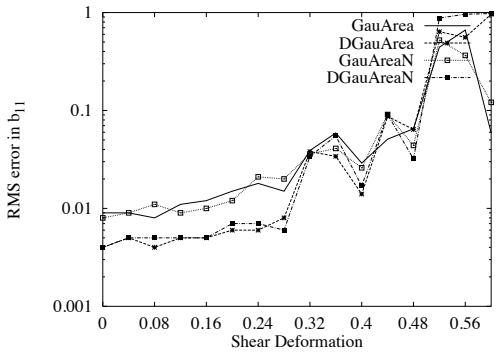



(e) RMS error in $t_x$

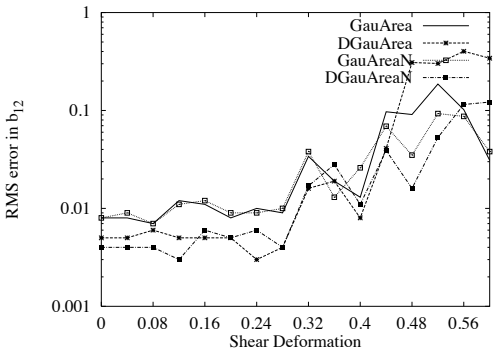



(f) RMS error in $t_y$

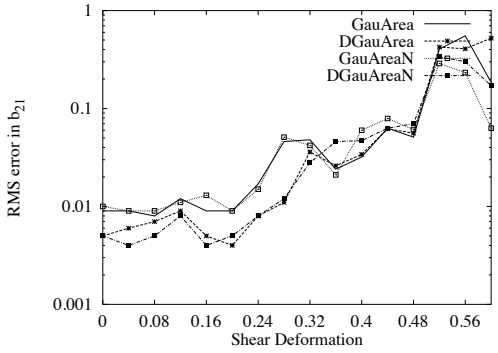
**Figure C.27.** Comparison of RMS errors for four different algorithms under rotational transformations. The error in the affine parameters is plotted against rotation angle.



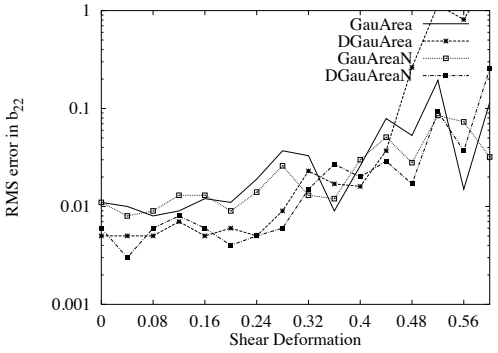

(a) RMS error in $b_{11}$



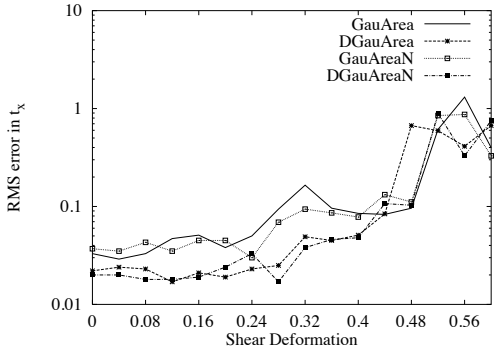

(b) RMS error in $b_{12}$



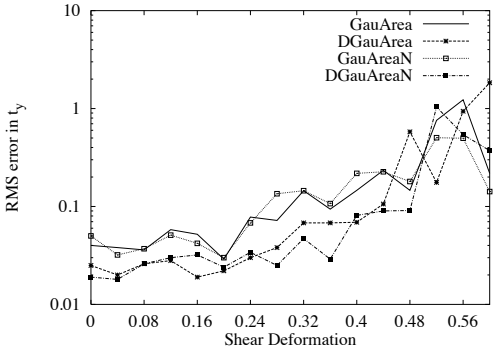

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$

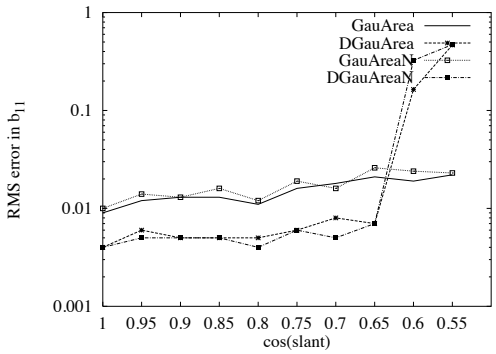



(e) RMS error in $t_x$

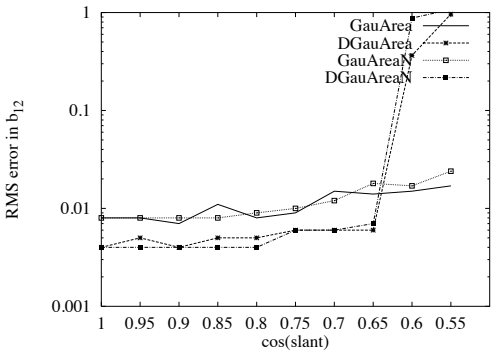



(f) RMS error in $t_y$

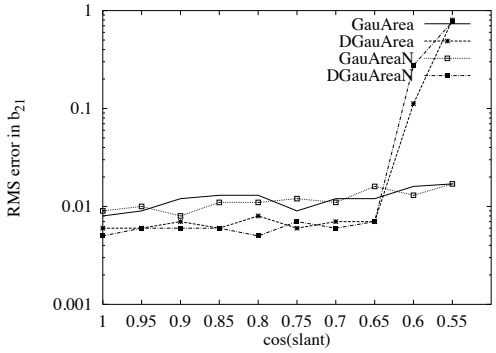
**Figure C.28.** Comparison of RMS errors for four different algorithms under shear deformations. The error in the affine parameters is plotted against shear deformation.



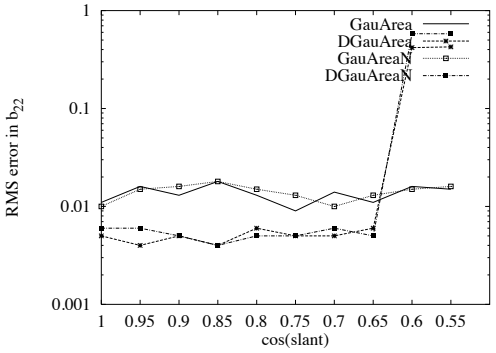

(a) RMS error in $b_{11}$



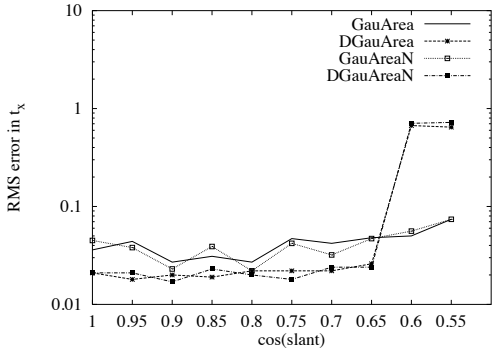

(b) RMS error in $b_{12}$



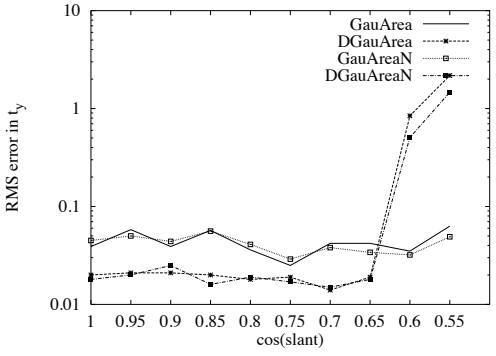

(c) RMS error in $b_{21}$




(d) RMS error in $b_{22}$




(e) RMS error in $t_x$




(f) RMS error in $t_y$

**Figure C.29.** Comparison RMS errors for four different algorithms for a plane moving in space. The error in the affine parameters is plotted against the cosine of the slant angle.

# BIBLIOGRAPHY


[1] Abu-Mostafa, Y.S., and Psaltis, D. Recognitive aspects of moment invariants. *IEEE Transactions on Pattern Analysis and Machine Intelligence 6* (1984), 698–706.

[2] Aloimonos, J. Y., and Swain, M. Shape from patterns: Regularization. *International Journal of Computer Vision 2* (1988), 171–187.

[3] Anandan, P. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision 2*, 3 (1989), 283–310.

[4] Anandan, P., Hanna, K., and Kumar, R. Shape recovery from multiple views: A parallax based approach. In *Proc. International Conference on Pattern Recognition* (1994), vol. A, pp. A:685–688.

[5] Bajcsy, R., and Lieberman, L. Texture gradient as a depth cue. *Computer Graphics and Image Processing* (1976), 52–67.

[6] Bergen, J.R., Anandan, P., Hanna, K. J., and Hingorani, R. Hierarchical model-based motion estimation. In *Proc. 2nd European Conference on Computer Vision* (1992), pp. 237–252.

[7] Blostein, D., and Ahuja, N. A multiscale region detector. *Computer Vision Graphics and Image Processing 45* (1989), 22–41.

[8] Bokser, M. Omnidocument technologies. *Proceedings IEEE 80*, 7 (1992), 1066–1078.

[9] Brown, L., and Shvaytser, H. Surface orientation from projective foreshortening of isotropic texture autocorrelation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 12*, 6 (1990), 584–588.

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

[11] Campani, M., and Verri, A. Computing optical flow from an overconstrained system of linear equations. In *Proc. 3nd Intl. Conf. on Computer Vision* (1990), pp. 22–26.

[12] Campani, M., and Verri, A. Motion analysis from optical flow. *Computer Vision Graphics and Image Processing:Image Understanding 56*, 12 (1992), 90–107.

[13] Cipolla, R., and Blake, A. Surface orientation and time to contact from image divergence and deformation. In *Proc. 2nd European Conference on Computer Vision* (1992), pp. 187–202.

[14] Cyganski, D., and Orr, J. A. Applications of tensor theory to object recognition and orientation determination. *IEEE Transactions on Pattern Analysis and Machine Intelligence 7* (1985), 662–673.

[15] Fennema, C. L., and Thompson, W. B. Velocity determination in scenes containing several moving objects. *Computer Graphics and Image Processing 9* (1979), 301–315.

[16] Florack, Ludvicus Maria Jozef. *The Syntactic Structure of Scalar Images*. PhD thesis, University of Utrecht, 1993.

[17] Garding, J. *Shape from Surface Markings*. PhD thesis, KTH, Stockholm, 1990.

[18] Gold, S., and Rangarajan, A. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18*, 4 (1996), 377–388.

[19] Haykin, S. *Communication Systems*. Wiley, 1978.

[20] Hildreth, E. C. *The Measurement of Visual Motion*. The MIT Press, Cambridge, MA, 1984.

[21] Horn, B. K. P., and Schunck, B. Determining optical flow. *Artificial Intelligence 17* (1981), 185–203.

[22] Hu, M.-K. Visual pattern recognition by moment invariants. *IRE Transactions of Information Theory IT-8* (1962), 179–187.

[23] Hummel, R., and Lowe, D. Computational considerations in convolution and feature extraction in images. In *From Pixels to Features*, J. C. Simon, Ed. Elsevier Science Publications B. V. (North-Holland), 1989, pp. 91–102.

[24] Huttenlocher, Daniel P., and Ullman, Shimon. Recognizing solid objects by alignment. In *Proc. Computer Vision and Pattern Recognition Conference* (1989), pp. 1114–1124.

[25] Irani, M., Anandan, P., and Hsu, S. Mosaic based representations of video sequences and their applications. In *Proc. 5th Intl. Conf. on Computer Vision* (June 1995), pp. 605–611.

[26] Jau, Y.C., and Chin, R.T. Shape from texture using the wigner distribution. In *Proc. Computer Vision and Pattern Recognition Conference* (1988), pp. 515–523.

[27] Jones, D. G., and Malik, J. A computational framework for determining stereo correspondence from a set of linear spatial filters. In *Proc. 2nd European Conference on Computer Vision* (1992), pp. 395–410.

[28] Jones, D. G., and Malik, J. Determining three-dimensional shape from orientation and spatial frequency disparities. In *Proc. 2nd European Conference on Computer Vision* (1992), pp. 661–669.

[29] Jones, G. J. F., Foote, J. T., Jones, K. Sparck, and Young, S. J. Video mail retrieval: The effect of word spotting accuracy on precision. In *International Conference on Acoustics, Speech and Signal Processing* (1995), vol. 1, pp. 309–316.

[30] Kanade, T., and Kender, J. R. Mapping image properties into shape constraints: Skewed symmetry, affine–transformable patterns, and the shape–from–texture paradigm. In *Human and Machine Vision*, J. Beck et al, Ed. Academic Press, NY, 1983, pp. 237–257.

[31] Kass, M. Linear image features in stereopsis. *International Journal of Computer Vision 1* (1988), 357–368.

[32] Khoubyari, Siamak, and J.Hull, Jonathan. Keyword location in noisy document image. In *Second Annual Symposium on Document Analysis and Information Retrieval, UNLV, Las Vegas* (1993), pp. 217–231.

[33] Koenderink, J. J. The structure of images. *Biological Cybernetics 50* (1984), 363–396.

[34] Koenderink, J. J., and van Doorn, A. J. Representation of local geometry in the visual system. *Biological Cybernetics 55* (1987), 367–375.

[35] Lee, Morris. Recovering the affine transformation of images by using moments and invariant axes. In *Proc. Topical Meeting of the Optical Society of America on Image Understanding* (1991).

[36] Lindeberg, T., and Garding, J. Shape from texture from a multi-scale perspective. In *Proc. 4th Intl. Conf. on Computer Vision* (1993), pp. 683–691.

[37] Lindeberg, Tony. *Scale-Space Theory in Computer Vision*. Kluwer Academic Publishers, 1994.

[38] Lu, C.P., and Mjolsness, E. Two-dimensional object localization by coarse to fine correlation matching. In *NIPS* (1994), pp. 985–992.

[39] Malik, J., and Rosenholtz, R. A differential method for computing local shape from texture for planar and curved surfaces. In *Proc. Computer Vision and Pattern Recognition Conference* (1993), pp. 267–273.

[40] Manmatha, R. Image matching under affine deformations. In *Proc. of the 27nd Asilomar IEEE Conf. on Signals, Systems and Computers* (1993), pp. 106–110.

[41] Manmatha, R. A framework for recovering affine transforms using points, lines or image brightnesses. In *Proc. Computer Vision and Pattern Recognition Conference* (1994), pp. 141–146.

[42] Manmatha, R., and Croft, W. B. Word spotting: Indexing handwritten manuscripts. In *Intelligent Multi-media Information Retrieval*, Mark Maybury, Ed. AAAI/MIT Press, April 1998.

[43] Manmatha, R., Han, Chengfeng, and Riseman, E. M. Word spotting: A new approach to indexing handwriting. Tech. Rep. CS-UM-95-105, Computer Science Dept, University of Massachusetts at Amherst, MA, 1995.

[44] Manmatha, R., Han, Chengfeng, and Riseman, E. M. Word spotting: A new approach to indexing handwriting. In *Proc. Computer Vision and Pattern Recognition Conference* (1996), pp. 631–637.

[45] Manmatha, R., Han, Chengfeng, Riseman, E. M., and Croft, W. B. Indexing handwriting using word matching. In *Digital Libraries '96: 1st ACM International Conference on Digital Libraries* (1996), pp. 151–159.

[46] Manmatha, R., and Oliensis, J. Measuring the affine transform – i: Scale and rotation. Tech. Rep. *CMPSCI TR 92–74,University of Massachusetts at Amherst,MA,1992. Also in Proc. of the Darpa Image Understanding Workshop 1993.*

[47] Manmatha, R., and Ravela, S. A syntactic characterization of appearance and its application to image retrieval. In *Proceedings of the SPIE conf. on Human Vision and Electronic Imaging II* (San Jose, CA, Feb. 1997), vol. 3016.

[48] Marinos, C., and Blake, A. Shape from texture: the homogeneity hypothesis. In *Proc. 3nd Intl. Conf. on Computer Vision* (1990), pp. 350–353.

[49] Marr, D. *Vision*. W.H. Freeman: San Francisco, 1982.

[50] Mori, S., Suen, C. Y., and Yamamoto, K. Historical review of ocr research and development. *Proceedings of the IEEE 80*, 7 (July 1992), 1029–1058.

[51] Myles, Z., and Lobo, N. V. Recovering affine motion and defocus blur simultaneously. In *Proc. Computer Vision and Pattern Recognition Conference* (1996), pp. 756–763.

[52] Ohta, Y., Maenobu, K., and Sakai, T. Obtaining surface orientation from texels under perspective projection. In *Proc. International Joint Conference on Artificial Intelligence* (1981), pp. 746–751.

[53] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T. *Numerical Recipes in C*. Cambridge University Press, 1986.

[54] Ravela, S., and Manmatha, R. Image retrieval by appearance. In *Accepted to the 20th Intl. Conf. on Research and Development in Information Retrieval (SIGIR'97)* (July 1997).

[55] Reiss, T. H. *Recognizing Planar Objects Using Invariant Image Features*. Springer-Verlag, 1993.

[56] Sawhney, H. S., Ayer, S., and Gorkani, M. Model-based 2d and 3d dominant motion estimation for mosaicing and video representation. In *Proc. 5th Intl. Conf. on Computer Vision* (June 1995), pp. 583–590.

[57] Sawhney, H. S., and Hanson, A. R. Identification and 3D description of 'shallow' environmental structure in a sequence of images. In *Proc. Computer Vision and Pattern Recognition Conference* (1991), pp. 179–186.

[58] Sawhney, H.S. 3d geometry from planar parallax. In *Proc. Computer Vision and Pattern Recognition Conference* (1994), pp. 929–934.

[59] Sawhney, H.S. Trackability as a cue for potential obstacle identification and 3-d description. *International Journal of Computer Vision 11* (1994), 237–265.

[60] Schmid, C., and Mohr, R. Combining greyvalue invariants with local constraints for object recognition. In *Proc. Computer Vision and Pattern Recognition Conference* (1996), pp. 872–877.

[61] Scott, G. L., and Longuet-Higgins, H. C. An algorithm for associating the features of two patterns. *Proc. Royal Society of London B B244* (1991), 21–26.

[62] Segman, J., Rubinstein, J., and Zeevi, Y.Y. The canonical coordinates method for pattern deformation: Theoretical and computational considerations. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14*, 12 (1992), 1171–1183.

[63] Shapiro, L. S., and Brady, J. M. Feature-based correspondence: An eigenvector approach. *Image and Vision Computing 10* (1992), 283–288.

[64] Shi, J., and Tomasi, C. Good features to track. In *Proc. Computer Vision and Pattern Recognition Conference* (1994), pp. 593–600.

[65] Stone, J. V. Shape from local and global analysis of texture. *Phil. Trans. R. Soc. Lond. B 339* (1993), 53–65.

[66] Super, B. J., and Bovik, A.C. Three-dimensional orientation from texture using gabor wavelets. In *SPIE Conf. on Visual Communications and Image Processing* (1991).

[67] Super, B. J., and Bovik, A.C. Solution to shape-from-texture by wavelet-based measurement of local spectral moments. In *Proc. Computer Vision and Pattern Recognition Conference* (1992).

[68] Szeliski, R. Video mosaics for virtual environments. *IEEE Computer and Applications 16*, 2 (March 1996), 22–30.

[69] Szeliski, R., and Coughlan, J. Hierarchical spline-based image registration. In *Proc. Computer Vision and Pattern Recognition Conference* (1994), pp. 194–201.

[70] Turtle, H.R., and Croft, W.B. A comparison of text retrieval models. *Computer Journal 35*, 3 (1992), 279–290.

[71] Ullman, S. *High-level Vision: Object Recognition and Visual Cognition*. MIT Press, 1996.

[72] van Rijsbegen, C. J. *Information Retrieval*. Butterworths, 1979.

[73] Weber, J., and Malik, J. Robust computation of optical flow in a multi-scale differential framework. In *Proc. 4th Intl. Conf. on Computer Vision* (1993), pp. 12–120.

[74] Weng, J. Image matching using the windowed fourier phase. *International Journal of Computer Vision 11*, 3 (December 1993), 211–236.

[75] Werkhoven, P., and Koenderink, J. J. Extraction of motion parallax structure in the visual system I. *Biological Cybernetics* (1990).

[76] Werkhoven, P., and Koenderink, J. J. Extraction of motion parallax structure in the visual system II. *Biological Cybernetics* (1990).

[77] Witkin, A. P. Recovering surface shape and orientation from texture. *Artificial Intelligence* (1981), 17–46.

[78] Xiong, Y., and Shafer, S. A. Moment and hypergeometric filters for high precision computation of focus, stereo and optical flow. Tech. Rep. Technical Report No. CMU-RI-TR-94-28, Carnegie Mellon University, 1994.