Chapter 1

Efficient Transformation Estimation Using Lie Operators: Theory, Algorithms, and Computational Efficiencies

W. David Pan

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/53271

1. Introduction

In many pattern recognition problems such as handwritten character recognition, it would be a challenge to design a good classification function, which can eliminate irrelevant variabilities among objects of the same class, while at the same time, being able to identify meaningful differences between objects of different classes. For example, in order for an automatic technique to "recognize" a handwritten digit, the incoming digit pattern needs to be accurately classified into one out of ten possible categories (from "0" to "9"). One straightforward yet inefficient way of implementation would be to match the pattern with a set of prototypes, where almost all possible instances (e.g., different sizes, angles, skews, etc.) of the digit in each category must be stored, according to a certain distance measure. Consequently, the pattern will be classified into the category where the closest match with one of its prototype instances was found. This approach would lead to impractically large prototype sets in order to achieve high recognition accuracy. An alternative method is to use only one prototype for each category, where different "deformed" instances of the same prototype can be generated by geometric transformations (e.g., thickened or rotated) during the matching process so as to best fit the incoming digit pattern. To this end, the concept of *Lie operators* for the transformations would be applicable.

More precisely, the pixel values of an incoming pattern (an digital image with $N \times N$ pixels) can be viewed as the components of a N^2-dimensional (N^2-D) vector. One pattern, or one prototype, is a point in this N^2-D space. If we assume that the set of allowable transformations is continuous, then the set of all the patterns that can be obtained by transforming one prototype using one or a combination of allowable transformations is a surface in the N^2-D pixel space. For instance, when a pattern I is transformed (e.g., rotated by an angle θ) according to a transformation $s(I, \theta)$, where θ is the only parameter, then the set of all the transformed patterns

$$T_I = \{x | \exists \theta, \text{for which } x = s(I, \theta)\} \tag{1}$$

is a one-dimensional curve in the N^2-D space. Here we assume that s is differentiable with respect to both I and θ, and $s(I,0) = I$. When the set of transformations is parameterized by m parameters θ_i, where $i = 1,2,...,m$, T_I becomes a manifold (topological surface) with an intrinsic dimension being m. For instance, if the allowable transformations of character images are rotations and scaling, the surface will be a 2-D manifold.

In practice, the search for the best matching deformation of a prototype for an incoming pattern would be expensive computationally, if the set of all the patterns that can be obtained by transforming one prototype using one or a combination of allowable transformations is large. Therefore, computationally efficient transformation estimation methods for pattern matching will be highly desired. It turns out that the surface of possible transforms of a pattern can be approximated by its *tangent plane* at the pattern [18]. More specifically, a linear approximation to the transform $s(I,\theta)$ of the pattern I can be obtained by the Taylor expansion of s around $\theta = 0$:

$$s(I,\theta) = s(I,0) + \theta\frac{\partial s(I,\theta)}{\partial \theta} + O(\theta^2) \approx I + \theta L, \tag{2}$$

where $L = \frac{\partial s(I,\theta)}{\partial \theta}$ is called the Lie Derivative of the transform s, which is also known as the tangent vector.

To facilitate a better understanding of the key concepts of Lie derivatives, which establish a connection between groups of transformations of the input space and their effect on a functional of that space, as well as Lie operators, which can be used to approximate the transformed pattern in a computationally efficient way, we first provide an explanation of the theory in Section 2, by working through some concrete examples of Lie groups and algebras. We then address in Section 3 the key problem of transformation estimation where both fast and accurate estimation methods are desired. The computational efficiency of transformation estimation algorithms based on Lie operator based approach is then investigated in Section 4, where several fast search algorithms for transformation estimation in video coding are presented. Further investigation is conducted in Section 5, by comparing the Lie operator based approach against transformation estimation based on a full affine transform model, in terms of the tradeoffs between accuracies and computational efficiencies.

2. Theory

2.1. Lie groups

Being an algebraic structure, a group is a set with an operation that combines any two of its elements to form a third element. To qualify as a group, the set and the operation must satisfy four conditions, namely, closure, associativity, identity, and invertibility (see definition below). For instance, the integers endowed with the addition operation form a group.

Definition: A set with elements g_i, g_j, g_k, ..., together with a combinatorial operation $\circ$ form a group G if the following axioms are satisfied [5]:

(i) **Closure**: If $g_i \in G$ and $g_j \in G$, then $g_i \circ g_j \in G$.

(ii) **Associativity**: If $g_i \in G$, $g_j \in G$, and $g_k \in G$, then $(g_i \circ g_j) \circ g_k = g_i \circ (g_j \circ g_k)$.

(iii) **Identity**: There exists an element e such that for every element $g_i \in G$, we have

$$g_i \circ e = g_i = e \circ g_i.$$

(iv) **Inverse**: Every group element g_i has an inverse (called g_i^{-1}), with the property

$$g_i \circ g_i^{-1} = e = g_i^{-1} \circ g_i.$$

Some groups carry additional geometric structures. For example, Lie groups are groups that also have a smooth (differentiable) manifold structure. The circle and the sphere are examples of smooth manifolds. Named after Sophus Lie, a nineteenth century Norwegian mathematician who laid the foundations of the theory of continuous transformation groups, Lie groups lie at the intersection of two fundamental fields of mathematics: algebra and geometry. A Lie group has the property that the group operations are compatible with its smooth structure. That is, the group operations are differentiable. More precisely, we have

Definition: A Lie group consists of a manifold $\mathcal{M}^n$ that parameterizes the group elements $g(x), x \in \mathcal{M}^n$ and a combinatorial operation defined by $g(x) \circ g(y) = g(z)$, where the coordinate $z \in \mathcal{M}^n$ depends on the coordinates $x \in \mathcal{M}^n$, and $y \in \mathcal{M}^n$ through a function $z = \Phi(x, y)$. There are two topological axioms for a Lie group [5].

(i) **Smoothness of the group composition map**: The group composition map $z = \Phi(x, y)$ is differentiable.

(ii) **Smoothness of the group inversion map**: The group inversion map $y = \psi(x)$, defined by $g(x)^{-1} = g(y)$, is differentiable.

Almost every Lie group is either a matrix group or equivalent to a matrix group, which greatly simplifies the description of the algebraic, topological, and continuity properties of the Lie groups. Let us consider the following example encountered in pattern recognition, where a prototype pattern can be represented as a computer image $P[i, j]$, which can be interpreted as the discrete version of the continuous function $f(X) = f(x, y)$. Assume that f is a differential function that maps points $X = (x, y)$ in the plane $\Re^2$ to $\Re$, which is the intensity (or pixel value) of the point X.

$$f : X \in \Re^2 \mapsto f(X) \in \Re. \tag{3}$$

Next, the image is deformed (e.g., rotate by an angle θ)) via a transformation T_θ (parameterized by θ), which maps bijectively a point of $\Re^2$ back to a point of $\Re^2$:

$$T_\theta : X \in \Re^2 \mapsto T_\theta(X) \in \Re^2. \tag{4}$$

For example, T_θ could represent rotating the pattern by an angle θ:

$$T_\theta : (u, v) \mapsto (u \cos\theta - v \sin\theta, u \sin\theta + v \cos\theta). \tag{5}$$

These transformations form a group G, which can be represented by a matrix group, with the combinatorial operation $\circ$ being the matrix multiplication. In particular, each element $g(\theta)$ of G is parameterized by one parameter θ:

$$g(\theta) = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}. \tag{6}$$

We show that G is indeed a group:

(i) **Closure**: If $g(\theta_1) \in G$ and $g(\theta_2) \in G$, then

$$g(\theta_1) \circ g(\theta_2) = \begin{pmatrix} \cos\theta_2 & \sin\theta_2 \\ -\sin\theta_2 & \cos\theta_2 \end{pmatrix} \begin{pmatrix} \cos\theta_1 & \sin\theta_1 \\ -\sin\theta_1 & \cos\theta_1 \end{pmatrix} = \begin{pmatrix} \cos(\theta_1+\theta_2) & \sin(\theta_1+\theta_2) \\ -\sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) \end{pmatrix} = g(\theta_1+\theta_2) \in G. \tag{7}$$

(ii) **Associativity**: If $g(\theta_1) \in G$, $g(\theta_2) \in G$, and $g(\theta_3) \in G$, then

$$(g(\theta_1) \circ g(\theta_2)) \circ g(\theta_3) = g(\theta_1+\theta_2) \circ g(\theta_3) = g(\theta_1+\theta_2+\theta_3), \tag{8}$$

and

$$g(\theta_1) \circ (g(\theta_2) \circ g(\theta_3)) = g(\theta_1) \circ g(\theta_2+\theta_3) = g(\theta_1+\theta_2+\theta_3). \tag{9}$$

Thus

$$(g(\theta_1) \circ g(\theta_2)) \circ g(\theta_3) = g(\theta_1) \circ (g(\theta_2) \circ g(\theta_3)). \tag{10}$$

(iii) **Identity**: There exists an element $e = g(0) = I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ such that for every element $g(\theta) \in G$, we have

$$g(\theta) \circ e = g(\theta) = e \circ g_i.$$

(iv) **Inverse**: Every group element $g(\theta)$ has an inverse $g(\theta)^{-1} = g(-\theta)$, such that

$$g(\theta) \circ g(\theta)^{-1} = I_2 = e = g(\theta)^{-1} \circ g(\theta).$$

We further show that G is also a Lie group with one parameter. To verify the two topological axioms for a Lie group, consider the group elements $g(\theta_1)$, $g(\theta_2)$, and $g(\theta_3)$, which are parameterized by $\theta_i \in M$, where M is one-dimensional curve (a smooth manifold). Given the combinatorial operation $g(\theta_1) \circ g(\theta_2) = g(\theta_3)$, it follows that the group composition map

$$\theta_3(\theta_1, \theta_2) = \theta_1 + \theta_2 \tag{11}$$

is differentiable. Furthermore, given the inverse $g(\theta_1)^{-1} = g(\theta_2)$ the group inversion map

$$\theta_2(\theta_1) = -\theta_1 \tag{12}$$

is also differentiable.

The study of Lie groups can be greatly simplified by linearizing the group in the neighborhood of its identity. This results in a linear vector space called a *Lie algebra* [4]. The Lie algebra retains most of the properties of the original Lie group. Next, we use again the rotation of an image as an example of transformation to illustrate how to linearize the Lie transformation group.

2.2. Lie operators and Lie algebras

Assume that the intensity of the original 2D image at location (u, v) is given by $f(u, v)$, where f is a differentiable function. In order to determine the intensity of the rotated image at a point (x, y), we need to calculate the location from which the rotation operation originated. This can be accomplished by taking the inverse transformation as

$$T_\theta^{-1} : (x, y) \mapsto (x\cos\theta + y\sin\theta, -x\sin\theta + y\cos\theta). \tag{13}$$

Let $s(f,\theta)(x,y)$ denote the intensity of the rotated image at point (x,y), then

$$s(f,\theta)(x,y) = f(x\cos\theta + y\sin\theta, -x\sin\theta + y\cos\theta). \tag{14}$$

That is, the intensity of the rotated pattern at point (x,y) equals to the intensity of the original pattern at the coordinate found by applying T_θ^{-1} on (x,y). Differentiating s with respect to θ around $\theta = 0$ gives

$$\begin{aligned}
&\frac{\partial s(f,\theta)}{\partial\theta}(x,y)\bigg|_{\theta=0} \\
&= \frac{\partial f}{\partial x}(x,y)\cdot\frac{\partial}{\partial\theta}(x\cos\theta + y\sin\theta)\bigg|_{\theta=0} + \frac{\partial f}{\partial y}(x,y)\cdot\frac{\partial}{\partial\theta}(-x\sin\theta + y\cos\theta)\bigg|_{\theta=0} \\
&= y\frac{\partial f}{\partial x}(x,y) - x\frac{\partial f}{\partial y}(x,y)
\end{aligned} \tag{15}$$

Using Taylor series expansion, we have

$$s(f,\theta)(x,y) = f(x,y) + \theta\left[y\frac{\partial f}{\partial x}(x,y) - x\frac{\partial f}{\partial y}(x,y)\right] + o(\|\theta\|^2)f(x,y). \tag{16}$$

Thus the intensity of the rotated pattern image can be approximated by

$$s(f,\theta)(x,y) \approx f(x,y) + \theta\cdot L_\theta(f(x,y)), \tag{17}$$

where L_θ is the so-called *Lie operator*, given by

$$L_\theta = y\frac{\partial}{\partial x} - x\frac{\partial}{\partial y} \tag{18}$$

Each rotated image with a certain angle θ corresponds to a point from a Lie group with one parameter.

More generally, if the transformation group is a Lie group with m parameters $\Theta = (\theta_1,\theta_2,\ldots,\theta_m)$, then after transformation, the intensity of the deformed image, $s(f,\Theta)$ is related to the original image f by the following approximation:

$$s(f,\Theta) = f + \theta_1\cdot L_{\theta_1}(f) + \theta_2\cdot L_{\theta_2}(f) + \cdots + \theta_m\cdot L_{\theta_m}(f) + o(\|\Theta\|^2)(f), \tag{19}$$

where the operators $L_{\theta_1}, L_{\theta_2}, \cdots, L_{\theta_m}$ are said to generate a Lie algebra, which is a linear vector space. A vector space is a mathematical structure formed by a collection of vectors, which may be added together and multiplied by numbers (scalars). More precisely,

Definition: A **Lie algebra** is a vector space V over a field $\mathbb{F}$, with an product operation $V \times V \to V$ denoted by $[X,Y]$, which is called the Lie bracket of $X \in V$ and $Y \in V$, with the following axioms [16]:

(i) The bracket operation is bilinear.

(ii) $[X,X] = 0, \forall X \in V$.

(iii) **Jacobi identity**: $[X,[Y,Z]] + [Y,[Z,X]] + [Z,[X,Y]] = 0$. $(X,Y,Z \in V)$.

In axiom (i), the bilinear operation refers to a function that combining two elements of the vector space to yield a third element in the vector space, which is linear in each of its arguments. As an example, matrix multiplication is bilinear: $M_1(n,n)M_2(n,n) = M_3(n,n)$.

To illustrate the concept of Lie brackets, let us consider another transformation with three parameters (a,b,c)

$$T^{-1}_{(a,b,c)} : (x,y) \mapsto (ax+c, by), \tag{20}$$

which corresponds to the matrix group

$$g(a,b,c) = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ c & 0 & 1 \end{pmatrix}. \tag{21}$$

Similar to the group $g(\theta)$ in (6), it can be shown that $g(a,b,c)$ is also a Lie group. However, the intensity of the pattern image after this new transformation is given by

$$s(f,a,b,c)(x,y) = f(ax+c, by). \tag{22}$$

By following the procedure outlined in (15) through (18), we can obtain the three Lie operators as follows:

$$L_a = x\frac{\partial}{\partial x},\ L_b = y\frac{\partial}{\partial y}, \text{ and } L_c = \frac{\partial}{\partial x}. \tag{23}$$

These three Lie operators generate a Lie algebra, with the Lie bracket between any two operators X and Y defined as

$$[X,Y] = X \circ Y - Y \circ X, \tag{24}$$

where $X \circ Y$ denotes the operation of applying the operator Y, followed by applying the operator X.

It can be easily checked that the Lie bracket $[X,Y]$ is bilinear (axiom (i) of Lie algebra). Next, for any operator $X \in L_a, L_b, L_c$, we have $[X,X] = X \circ X - X \circ X = 0$, thereby satisfying axiom (ii). Verifying the Jacob identify requires additional efforts. First, we have

$$L_a \circ L_b = x\frac{\partial}{\partial x}\left(y\frac{\partial}{\partial y}\right) = xy\frac{\partial^2}{\partial x \partial y} = y\frac{\partial}{\partial y}\left(x\frac{\partial}{\partial x}\right) = L_b \circ L_a, \tag{25}$$

Hence

$$[L_a, L_b] = L_a \circ L_b - L_b \circ L_a = 0. \tag{26}$$

Similarly,

$$\begin{aligned}[L_a, L_c] &= L_a \circ L_c - L_c \circ L_a = x\frac{\partial}{\partial x}\left(\frac{\partial}{\partial x}\right) - \frac{\partial}{\partial x}\left(x\frac{\partial}{\partial x}\right) = x\frac{\partial^2}{\partial x^2} - \left(\frac{\partial}{\partial x} + x\frac{\partial^2}{\partial x^2}\right) \\ &= -\frac{\partial}{\partial x} = -L_c,\end{aligned} \tag{27}$$

and

$$[L_b, L_c] = L_b \circ L_c - L_c \circ L_b = y\frac{\partial}{\partial y}\left(\frac{\partial}{\partial x}\right) - \frac{\partial}{\partial x}\left(y\frac{\partial}{\partial y}\right) = y\frac{\partial^2}{\partial y \partial x} - y\frac{\partial^2}{\partial x \partial y} = 0. \tag{28}$$

Therefore,

$$[L_a,[L_b,L_c]]=[L_a,0]=0,\ [L_b,[L_c,L_a]]=[L_b,L_c]=0,\ \text{and}\ [L_c,[L_a,L_b]]=[L_c,0]=0. \quad (29)$$

It follows that the Jacob identity holds.

The result of applying the three Lie operators to a function f, which is a 2D image in our example, is the set of vectors known as *tangent vectors* (also called the *Lie derivatives* of the transformation). These tangent vectors generate the so-called *tangent space*. Each point in the tangent space corresponds to a transformation, and any transformation of the Lie group $g(a,b,c)$ corresponds to a point in the tangent space.

2.3. Lie operators on discrete images

As shown in (17), given a continuous image f, by applying the Lie operator (L_θ) for rotation, we can approximate the rotated image as $s(f,\theta)=f+\theta\cdot L_\theta(f)$. However, in many practical applications, we need to deal with computer images. Given a discrete image I, in order to apply a Lie operator, which involves derivatives, we first convert I into a continuous one (f) by means of convolution: $f=I*g_\sigma$, where g_σ is a 2D Gaussian function defined in [18] as:

$$g_\sigma=\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right). \quad (30)$$

In our study, besides rotation (R), we will consider several other types of transformations, such as scaling (S), parallel deformation (P), and diagonal deformation (D), as defined in Table 1. To distinguish the Lie operators for different types of transformations, we use L_R to denote the Lie operator for rotation. After applying $L_R=y\frac{\partial}{\partial x}-x\frac{\partial}{\partial y}$, we have

$$L_R(f)=\left(y\frac{\partial}{\partial x}-x\frac{\partial}{\partial y}\right)(I*g_\sigma)=y\left(I*\frac{\partial g_\sigma}{\partial x}\right)-x\left(I*\frac{\partial g_\sigma}{\partial y}\right). \quad (31)$$

To avoid high computational complexity associated with the convolution operation and the calculation of the partial derivatives of the Gaussian function in (30), we can apply the Lie operator on the discrete image directly, by using the following approximations [14].

$$L_R(f)\approx L_R(I)=\left(y\frac{\partial}{\partial x}-x\frac{\partial}{\partial y}\right)I=y\left(\frac{\partial I}{\partial x}\right)-x\left(\frac{\partial I}{\partial y}\right), \quad (32)$$

where

$$\frac{\partial I}{\partial x}\approx\frac{1}{2}\left[I(x+1,y)-I(x-1,y)\right],\ \text{and}\ \frac{\partial I}{\partial y}\approx\frac{1}{2}\left[I(x,y+1)-I(x,y-1)\right]. \quad (33)$$

After the Lie operator is applied, the rotated version of the image I can then be easily obtained as

$$I_R=I+\theta\times L_R(I). \quad (34)$$

For small angles (θ), the approximation tends to be reasonably good.

Similarly, we can obtain the transformed images for other types of transformations, based on their associated Lie operators (summarized in the third column of Table 1), which can be derived in a similar fashion to L_R.

Transformation	Transformation matrix T_θ (adapted from [18])	Lie operator and the transformed image
Rotation (R)	$\begin{pmatrix} \cos\theta, & -\sin\theta \\ \sin\theta, & \cos\theta \end{pmatrix}$	$L_R = y\frac{\partial}{\partial x} - x\frac{\partial}{\partial y}$ $I_R = I + \theta \times L_R(I)$
Scaling (S)	$\begin{pmatrix} 1+\theta, & 0 \\ 0, & 1+\theta \end{pmatrix}$	$L_S = x\frac{\partial}{\partial x} + y\frac{\partial}{\partial y}$ $I_S = I + \theta \times L_S(I)$
Scaling (S_x)	$\begin{pmatrix} 1+\theta, 0 \\ 0, & 1 \end{pmatrix}$	$L_{S_x} = x\frac{\partial}{\partial x}$ $I_{S_x} = I + \theta \times L_{S_x}(I)$
Scaling (S_y)	$\begin{pmatrix} 1, & 0 \\ 0, 1+\theta \end{pmatrix}$	$L_{S_y} = y\frac{\partial}{\partial y}$ $I_{S_y} = I + \theta \times L_{S_y}(I)$
Parallel Deformation (P)	$\begin{pmatrix} 1+\theta, & 0 \\ 0, & 1-\theta \end{pmatrix}$	$L_P = x\frac{\partial}{\partial x} - y\frac{\partial}{\partial y}$ $I_P = I + \theta \times L_P(I)$
Diagonal Deformation (D)	$\begin{pmatrix} 1, \theta \\ \theta, 1 \end{pmatrix}$	$L_D = y\frac{\partial}{\partial x} + x\frac{\partial}{\partial y}$ $I_D = I + \theta \times L_D(I)$

Table 1. Six types of transformation and their associated Lie operators (θ is the degree of the transformations).

We can see from (32) that only simple subtractions and multiplications are involved in applying the Lie operator to obtain $L_R(I)$, which needs to be calculated just once, since a different transformed version I_R corresponding to a different degree of transformation (θ) can be obtained by using the same $L_R(I)$. Therefore, the implementation of Lie operators has fairly low computational complexity.

2.4. Lie operators for transformation estimation

Lie operators were proposed in [18] as an effective method for handling transformation invariance in handwritten digit pattern recognition [19]. In order for an automatic method to "recognize" a handwritten digit, the incoming digit pattern needs to be accurately classified into one out of ten possible categories. one method is to use an only one prototype image (I) for each category, with different "deformed" instances, $s(I, \Theta)$, of the same prototype image being generated by geometric transformations during the matching process so as to best fit the incoming digit pattern. As mentioned in the section 1, when the set of transformations is parameterized by m parameters $\theta_i \in \Theta$ (rotation, scaling, etc.), the transformed image $s(I, \Theta)$ is a surface (manifold) with intrinsic dimension of at most m. In general, such a manifold is not linear. Matching a deformable prototype to an incoming pattern now amounts to finding the point on the manifold that is at a minimum distance from the point in the pixel space corresponding to the incoming pattern. Because the manifold has no analytical expression, the matching process can be very difficult. However, if the set of transformations happens to be linear in the pixel space, then the manifold is a linear subspace (a plane). The matching procedure is then reduced to finding the shortest distance between a point (vector) and a plane, or between two tangent planes corresponding to their original manifolds, which is the idea of *tangent distance* in [18]. While the tangent distance is able to capture the transformation invariance, it involves solving of a complicated least-square problem, which is not only computationally expensive, but also prone to numerical instability issues associated with solving linear systems. Therefore, conventional Euclidean distance between patterns,

due to its fast and easy calculation, was also used in conjunction with the tangent distance in actual implementation.

On the other hand, for many pattern recognition tasks, e.g., character recognition, a set of allowable deformations of the prototype might have been known *a priori*. Therefore, one can generate on-the-fly a set of varying transformed versions of the same prototype I, by using the Lie operators associated with the transforms, in a computationally efficient way. For example, a set of rotated images $I_R(\theta_i)$, where $i = 1, 2, \ldots, n$, can be readily obtained by

$$I_R(\theta_i) = I + \theta_i \times L_R(I), \tag{35}$$

where $L_R(I)$ can be pre-computed and shared by calculations of different $I_R(\theta_i)$.

Thus, *transformation estimation* refers to matching an incoming pattern image P to the "closest" $I_R(\theta_i)$, which has the shortest distance with P. For simplicity, the Euclidean distance could be used.

In transformation estimation, we search for a value for θ that best matches the degree of transformation the prototype has undergone in relation to the incoming pattern. If the best θ value is found to be zero in the case of rotation, then the resultant rotated version will be the same as the original prototype. If θ has a larger search range, then the probability of finding a better match may be increased; however, the complexity of searching will be increased as we have to examine more candidates. On the other hand, the step size of θ is also an important parameter that controls the "granularity" of the searching. By decreasing the step size, we may be able to enlarge the searching range of θ without increasing the search complexity. We can further lower the searching complexity by using variable step sizes. For example, we can employ finer-granular searching by taking smaller step sizes for small θ values, whereas the step size increases as the search drifts away from the centers of the range of allowable θ values [11].

Transformation estimation can be viewed as a generalized operation of the translation motion estimation. In the following, we present a case study to illustrate the design of computationally efficient transformation estimation algorithms based on Lie operators, by selecting the subject of local motion estimation in video coding, where both accurate and fast motion estimation is critical [12]. We then discuss several methods in which multiple Lie operators can be combined to detect smaller degrees of object motions in video frames such as scaling, rotations and deformations, with varying computational complexities. We then provide both analytical and empirically obtained results regarding the tradeoffs between estimation accuracies and computational complexities for these methods [13].

3. Transformation estimation in video coding

Motion estimation is a critical component of almost every video coding system [10][17][23]. Most compression techniques exploit the temporal redundancy that exists between the succeeding frames. In motion estimation, we search for any object in the previous frame that provides a good match of an object in the current frame within a sequence of images (frames). Motion compensation refers to representing objects in the current frame by their match objects in the previous frame. Conventional motion estimation algorithms in video coding consider

only translations as an approximation to a combination of potential motions of objects in a video scene, including scaling, rotation, deformations and so on.

3.1. Block-based translation motion estimation

Block-based motion estimation [10][23] has been adopted in international standards for video coding such as MPEGs and H.264, where each frame is partitioned evenly into square blocks. Motion estimation is applied on a block-by-block basis so that each block is associated with a motion vector. Motion vectors are used to produce a motion-compensated prediction of a frame to be transmitted from a previously transmitted reference frame [1][7]. Motion estimation enables us to transmit the frame difference as an update between the current frame and the motion-compensated prediction of the current frame from the previous frame, rather than the entire current frame, thereby achieving compression by user fewer bits to code the current frame.

In block-based motion estimation, each frame is divided into evenly partitioned square blocks (4×4, 8×8, ..., etc.). We attempt to predict the current frame (F_2) from the previous frame (F_1). The prediction is obtained by taking the best match of each block of F_2 within the searching window of F_1. The match criterion is typically based on mean square error (MSE). The block with the minimum MSE is considered to be the best match, and its associated motion vector (dx, dy) is given by

$$(dx, dy) = \arg \min_{(du,dv)\in[-R,R]} \{\text{MSE}_{m,n} = \sum_{i,j=0}^{B-1} [(F_2(x, y) - F_1(x + du, y + dv)]^2\}, \tag{36}$$

where B is the block size, $[-R, R]$ is the searching window, and $x = m \times B + i$, and $y = n \times B + j$ for the block (m, n). Note that the motion vector for a still block is $(0, 0)$.

After finding in F_1 the best match block of each block in F_2, the prediction frame (P_1) of F_2 can then be constructed. To determine the accuracy of the prediction, the PSNR between F_2 and $P1$ is calculated as

$$\text{PSNR} = 10 \log_{10} \frac{255^2}{\text{MSE}_{avg}}, \tag{37}$$

where MSE_{avg} is the average mean square error between F_2 and P_1 as given by

$$\text{MSE}_{avg} = \sum_{m=0}^{M} \sum_{n=0}^{N} \frac{\text{MSE}_{m,n}}{M \times N}. \tag{38}$$

Note that $\text{MSE}_{m,n}$ is defined in (36), and $M \times N$ is the total number of blocks in a frame.

Conventional motion estimation algorithms in video coding consider only translations as an approximation to a variety of object motions; therefore, they have limitations in capturing potential motions such as scaling, rotations and deformations in a video scene other than the translation. The reason for the widespread use of the translation model lies partly in its simplicity - translation model can be readily characterized by displacement motion vectors and can thus be implemented with much lower complexity than other non-linear motion models used to describe non-translation motions. Nonetheless, the accuracy of the motion estimation would be sacrificed by considering the translation model alone.

3.2. Block-based transformation estimation

Non-translational transformation estimation can be introduced into video coding to further increase the overall motion estimation accuracy. More specifically, the conventional (translation) motion estimation is applied on the previous frame (F_1) based on the current frame (F_2). We can construct a predicted frame P (of the current frame) from the previous frame by using the resulting motion vectors associated with each block in the current frame (see Fig. 4). The accuracy of the predicted frame P (relative to the current frame F_2) can be represented by PSNR_1. Next, transformation estimation based on the Lie operators is applied on the match blocks (B_P) in the predicted frame P to further improve the motion estimation accuracy. For each block B_P in P, we search for the best parameter θ from the set of candidate parameters that yields the smallest mean square error between the transformed version B_T and the corresponding block (B_C) in the current frame F_2. Consequently, a new predicted frame P_T can be formed by the resulting blocks of B_T. The accuracy of the newly predicted frame P_T can be represented as PSNR_2. As expected, P_T will become a better prediction of the current frame than P, thereby achieving an increased accuracy in motion estimation and prediction. The accuracy of the motion estimation can be measured by the PSNR between the

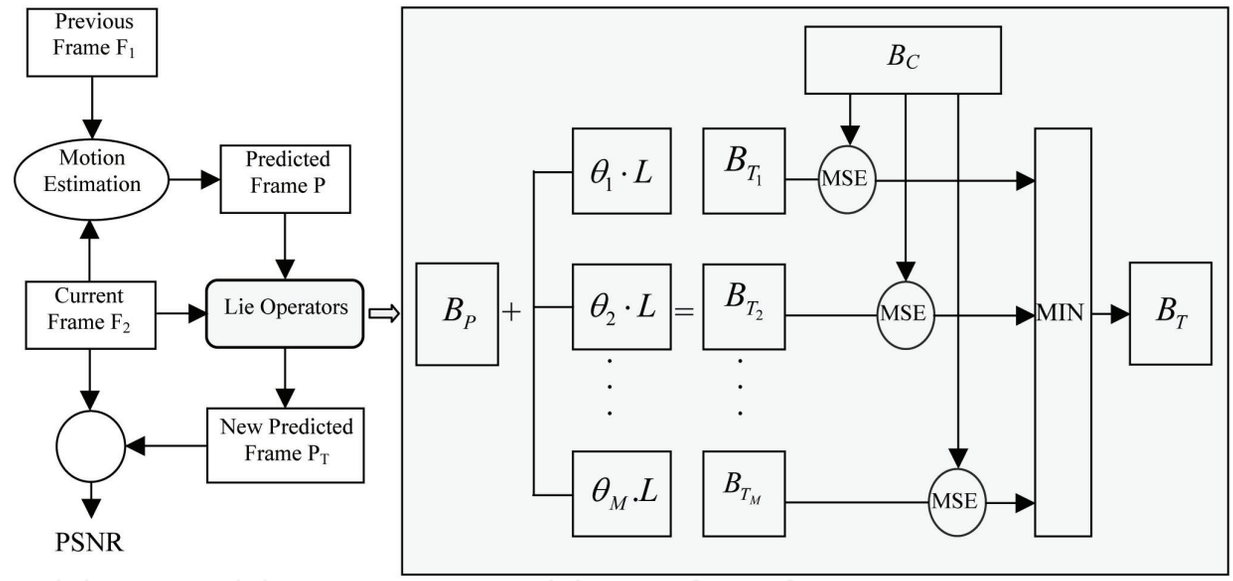

Figure 1. The transformation estimation system using a Lie operator: We search for the best θ in the set of candidates $[\theta_1, \theta_2, \ldots, \theta_M]$ such that the transformed block B_T of the block B_P in the prediction frame P will have the smallest MSE compared to the corresponding block B_C in F_2.

current frame and the predicted frame. The improved accuracy due to the motion models is calculated as $(\text{PSNR}_2 - \text{PSNR}_1)$. The accuracy of the motion estimation can be improved by considering other types of transformations as well.

4. Computational efficiencies of transformation estimation using multiple Lie operators

We first examine the full search method that exhaustively searches for the best combination of four types of Lie operators (R, S, P and D). In order to reduce the high computational complexity associated with the full search method, we then consider the following three

parameter-search methods: *dynamic programming* (DP)-like search, *iterative* search, and *serial* search. They combine the Lie operators in different ways, with varying accuracy-complexity tradeoffs [13].

4.1. Full search

Fig. 2 illustrates all possible combinations of the Lie operators for rotation, scaling, parallel deformation, and diagonal deformation. The highlighted path shows one combination ($D \rightarrow P \rightarrow S \rightarrow R$), which means that block B_P will be first diagonally deformed, and then the deformed block will go through the parallel deformation. The resultant block will be scaled and then rotated to obtain B_T. The degree of motions (θ) associated with each participating operator is optimized by the searching procedure illustrated in Fig. 1. The full search is the

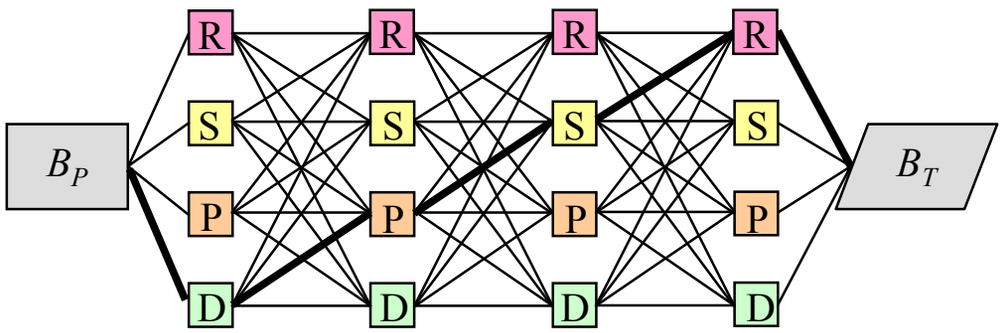

Figure 2. The trellis structure of the combined Lie operators. The output block B_T is expected to provide more accurate prediction than the input block B_P.

most straightforward and yet the most computationally expensive method. In this method, we search through all possible ($4^4 = 256$) paths that start from block B_P (of the predicted frame P) and end on the transformed block B_T (of a more accurately predicted frame than P), and select the path (i.e., the combination of the four Lie operators) whose output block B_T is the most accurately predicted version of block B_C in the current frame.

Assume that x is the computational complexity of motion estimation for a single Lie operator. Thus the complexity associated with any path of four operators from B_P to B_T in Fig. 2 is $4x$. Since we need to search all 256 possible paths, the overall complexity of the full search method will be $1024x$.

We can reduce the complexity of this brute-force search approach by dividing the estimation process into four stages, with each stage corresponding to one column of operators in Fig. 2. In the first stage, there will be four estimation operations for R, S, P, and D, respectively, with complexity being $4x$. In the second stage, we will apply the same four operators on one of the four candidate transformed blocks generated by one of the four operators in stage one. For example, starting with R in the first stage, we will examine $R \rightarrow R$ (R in the first stage, followed by R in the second stage), $R \rightarrow S$, $R \rightarrow P$, and $R \rightarrow D$. Note that applying the R operator again on a block already rotated by the best θ value as found in the first stage of estimation would not be beneficial in general. However, further gains in the estimation accuracy might be achievable by considering other combinations such as $R \rightarrow S$, $R \rightarrow P$, and $R \rightarrow D$. Therefore, the total complexity of the second stage will be $4 \times 4x = 16x$. Likewise, the complexity of the third stage will be $4 \times 16x = 64x$. In the last stage, the complexity will amount to $4 \times 64x = 256x$. Therefore, the overall complexity of the reduced-complexity full

search method is $340x$ ($= 4x + 16x + 64x + 256x$), merely $1/3$ of that of the brute-force full search method. Even so, the complexity of the full search is still unacceptably high in practical applications. In order to further reduce the complexity, let us consider the following search methods.

4.2. Dynamic-programming-like search

Compared to the full search, the DP search method is a sub-optimal search method, which has a flavor of the dynamic-programming (DP) solution in finding the shortest path through a weighted graph [2]. Similar to the Viterbi algorithm used in the decoding of convolutional codes [9], the DP search method keeps only those "survivors" (i.e., the best result obtained by each operator) in each stage (of the four stages in Fig. 2) for further searching operations. In the first stage, there are four transformed blocks ("survivors"), corresponding to the four Lie operators considered, as a result of the estimation operations (with complexity being 4x). In the second stage, four operators will be again applied to the survivors of stage 1. Take operator R as an example. Out of the four possible partial paths ($R \rightarrow R$, $S \rightarrow R$, $P \rightarrow R$, and $D \rightarrow R$) entering into R in the second stage, we choose the one that gives a transformed version with the smallest MSE value (as compared against the block in the current frame to be predicted). The transformed block so obtained will be stored as the survivor for operator R in the second stage, so will its originating operator in the first stage. Information about all other inferior partial paths will be discarded. In order to obtain other survivor blocks for stage two, the same procedure will be repeated for the other three operators. Therefore, the total complexity for stage two will be $4 \times 4x = 16x$. The four surviving blocks obtained in stage two will then be used for obtaining another four survivors in stage three in the same fashion, with the partial paths leading to the survivors getting longer. The same procedure will be repeated for stage four in order to obtain yet another four survivors, one of which with the least MSE will be the final winner. Hence the overall complexity for the DP-like method is $52x$ ($= 4x + 3 \times 16x$), which is less than $1/6$ of that of the reduced-complexity full search method. Although there is no guarantee of optimality in theory for this DP-like method, we expect its search results to be reasonably close to those yielded by the full search method.

4.3. Iterative search

To further reduce computational complexity, we introduce the iterative search method that performs the motion-parameter estimation through multiple iterations. In each iteration, we choose the best Lie operator (Fig. 3). For example, in the first iteration, the S operator may turn out to be the best operator. The scaled block will go through the same estimation process in the next iteration, which will output a transformed block with lower MSE values. Here we consider only four iterations to ensure fair comparison between this method and the other two methods previously discussed. Therefore, the overall complexity of this method will be $4 \times 4x = 16x$, which is slightly less than $1/3$ of the complexity of the DP-like method.

4.4. Serial search

In the foregoing two search methods, the best quadruplet of Lie operators will not be known until the search is completed. Their complexities are higher than a simplified search method, where Lie operators are applied sequentially in a pre-determined order (e.g., the order of $R \rightarrow S \rightarrow P \rightarrow D$ in Fig. 4). Although this serial search method has the lowest complexity ($4x$), it

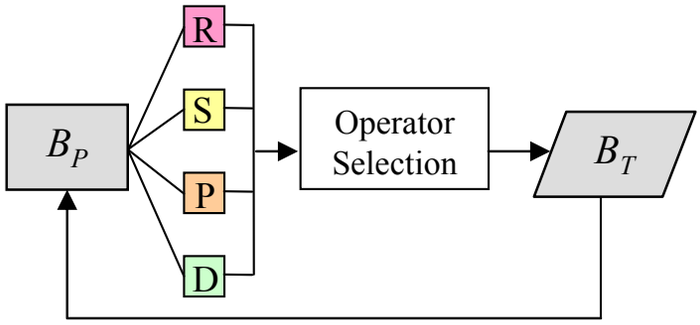

Figure 3. Iterative search. In each iteration, the best operator is selected as the one with the largest MSE reduction on the input block B_P. The transformed block B_T generated by the best operator found will be further transformed optimally in the next iteration.

is unlikely to provide very accurate transformation estimation due to the non-communicative nature of the transformations.

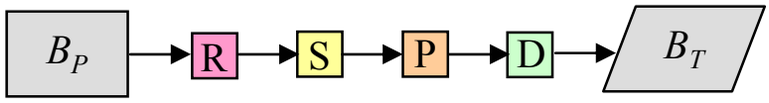

Figure 4. Serial search: we apply R, S, P and D operators sequentially to obtain the transformed block B_T.

4.5. Comparison of computational complexity

As summarized in Table 2, the complexity of the DP-like search is 13 times that of the serial search and the complexity of the iterative search is 4 times that of the serial search.

Search Method	Complexity
DP-like	$52x$
Iterative	$16x$
Serial	$4x$

Table 2. Complexities required by the three search methods (x is the complexity associated with estimation using an individual operator). θ was chosen from $[-0.14, 0.14]$, with a step size of 0.02.

4.6. Simulation results

We tested the above mentioned methods on three standard video sequences "Table Tennis", "Mobile Calendar", and "Tempete", all in the CIF format (288×352). Some samples frames of these sequences are shown in Figure 6.

In the simulations, a block size of (4×4) was used. The size of the search window is chosen to be ± 15 pixels in translation motion estimation that precedes the transformation estimation using Lie operators. The search range of ± 0.14 (with step size being 0.02) is chosen for θ_R, θ_S, θ_P, and θ_D for the DP-like, iterative, and serial search methods.

Simulation results are illustrated in Fig. 6, Fig. 7, and Fig. 8 for the three test sequences. Some statistics (maximum, minimum and the average values) of the PSNR improvements effected by the three search methods are listed in Table 3. We can see in Table 3 that the DP-like method significantly increases the accuracy of the predicted frames of all three sequences by as high

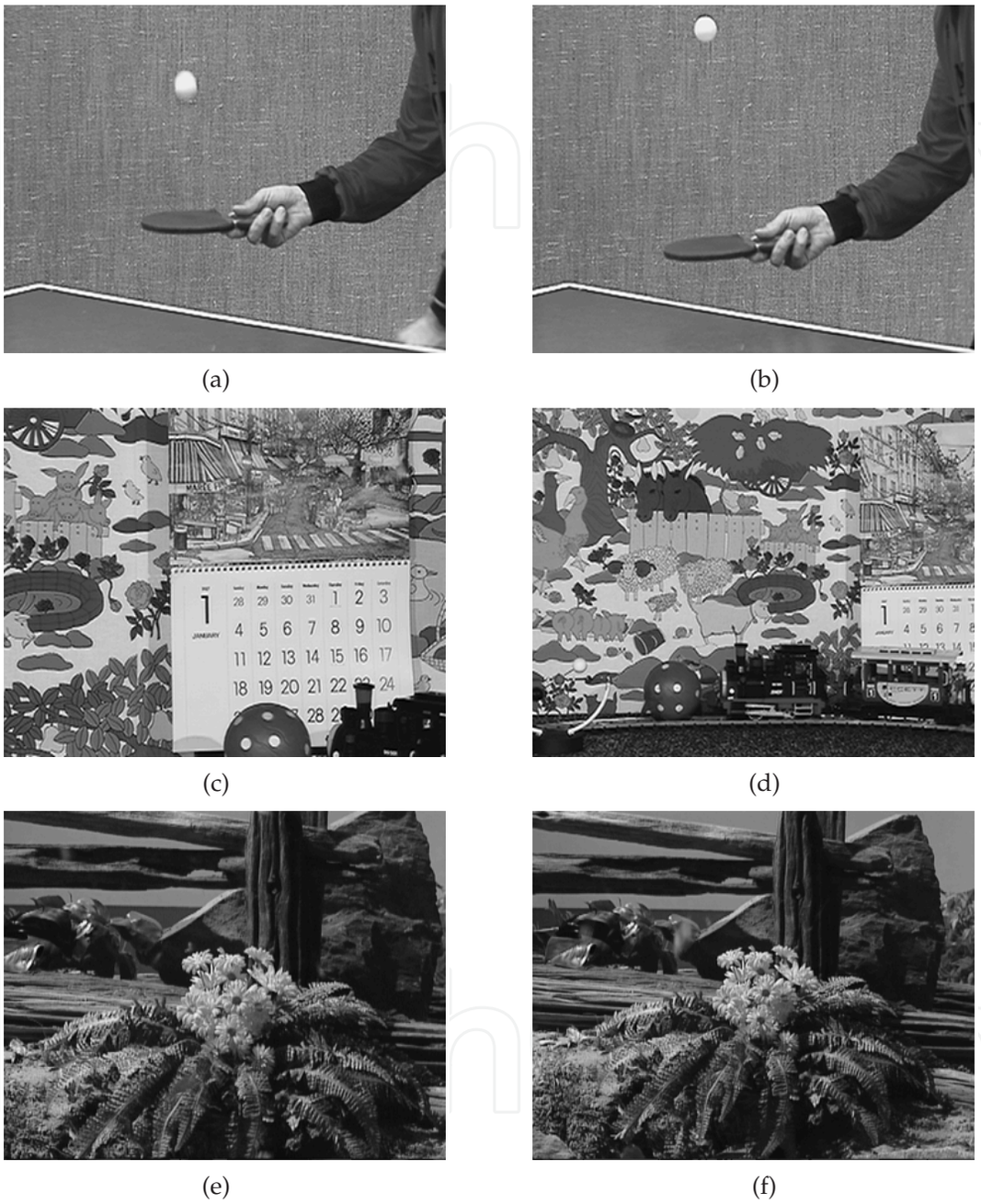

Figure 5. Sample frames of the video sequences. (a) The 1st frame of the "Table Tennis" sequence. (b) The 20th frame of the "Table Tennis" sequence. (c) The 1st frame of the "Mobile Calendar" sequence. (d) The 200th frame of the "Mobile Calendar" sequence. (e) The 1st frame of the "Tempete" sequence. (f) The 50th frame of the "Tempete" sequence.

as 2.6 dB and above 2.1 dB on average. The largest improvement (2.47 dB on average) is observed in "Mobile Calendar". This may be attributed to the existence of a great deal of non-translational motions in "Mobile Calendar" (e.g., the ball keeps rotating, and the camera is zooming out). On the other two sequences, about 2.1 dB increase can be achieved by the DP-like method.

Search Method	Table Tennis			Mobile Calendar			Tempete		
	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg
DP-like	2.67	0.68	2.19	2.65	2.15	2.47	2.30	1.46	2.12
Iterative	2.12	0.45	1.75	2.31	1.73	2.04	1.81	1.19	1.66
Serial	1.70	0.30	1.35	1.84	1.36	1.60	1.39	0.91	1.27

Table 3. Increased estimation accuracy (in dB) for the three video sequences.

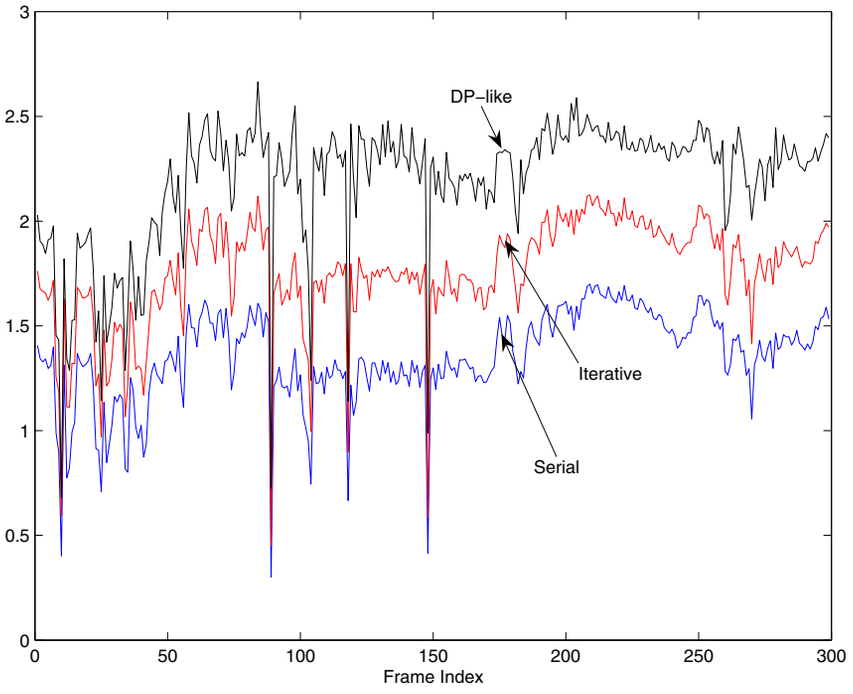

Figure 6. PSNR improvements (in dB) for "Tennis".

With less than 1/3 of the complexity required by the DP-like method, the iterative search can deliver an impressive estimation accuracy, especially on the "Mobile Calendar" (up to 2.31 dB and about 2dB on average). Similar to the case with the DP-like method, slightly lower PSNR improvements are observed on the other two sequences: on average, 1.75 dB and 1.66 dB for the sequences "Table Tennis" and "Tempete", respectively. As can be observed in Fig. 6, numerous deep plunges of the PSNR improvement (occurring in a range of frames around, e.g., 90 and 149) affect adversely the average PSNR improvement for "Table Tennis". These plunges occur whenever there is a scene change. For "Tempete", although there is no major scene change, a continuous influx of large number of new objects (e.g., small leaves blown by wind) tends to make transformation estimation less effective.

With only one quarter of the complexity required by the iterative search, the serial search achieves average PSNR improvements of 1.60dB, 1.35dB and 1.27 dB on "Mobile Calendar", "Table Tennis", and "Tempete", respectively. The accuracy of this method is the lowest, which

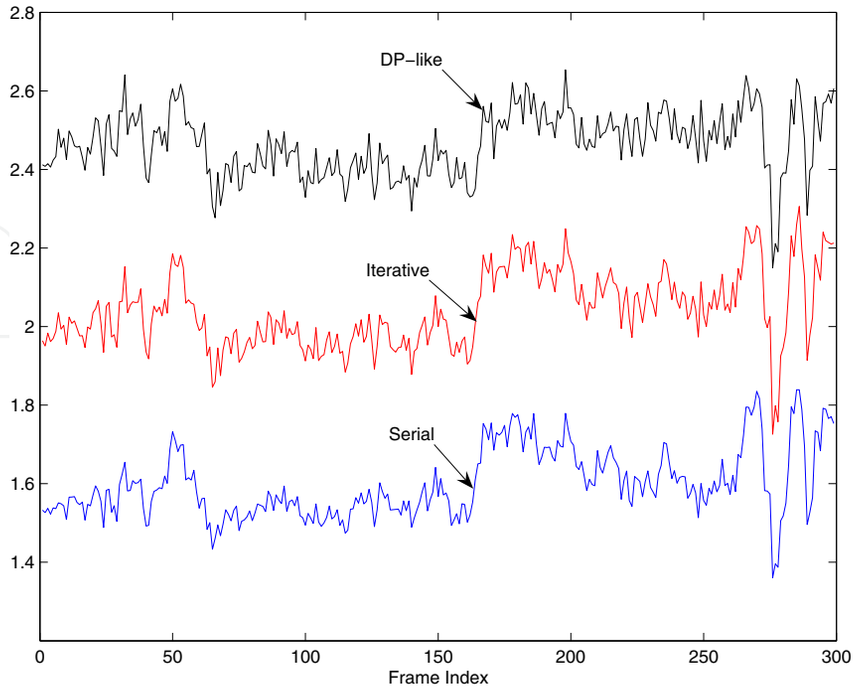

Figure 7. PSNR improvements (in dB) for "Mobile Calendar".

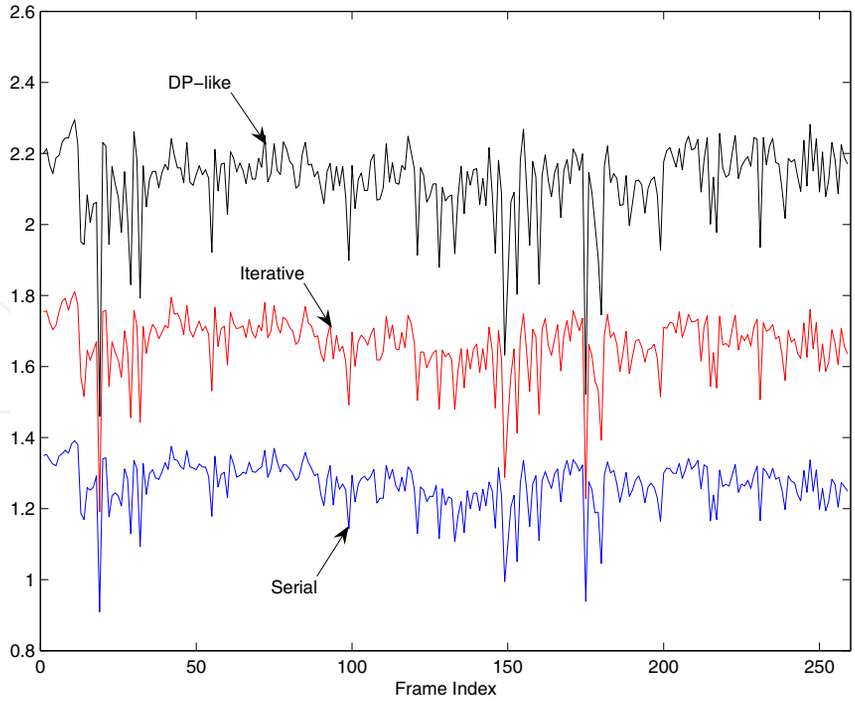

Figure 8. PSNR improvements (in dB) for "Tempete".

indicates that changing the order of the Lie operators in a sequence does affects the motion estimation accuracy.

We also measured the actual computation times of the three search methods on a PC running Windows XP (with 3.40 GHz Pentium 4 CPU and 2GB RAM). The total running time of the subroutine for each method was first measured over all the frames in a test sequence. Then the average running time per block for each search method was calculated and listed in Table 4. On average, Time (DP like search) / Time (Serial Search) = 13.12, and Time (Iterative) / Time (Serial Search) = 4.03, which is in agreement with the analytical results listed in Table 2. As a reference, the average time was also measured for executing the subroutine for the conventional translation-only motion estimation that precedes the transformation estimation. As shown in Table 4, the complexity of the DP-like search, iterative search and the serial search methods is 69%, 21% and 5%, respectively, relative to that of the translation-only motion estimation method.

Search Method	Table Tennis	Mobile Calendar	Tempete	Average	Normalized Complexity
DP-like	2.469	2.500	2.470	2.480	0.69
Iterative	0.763	0.766	0.756	0.762	0.21
Serial	0.181	0.195	0.192	0.189	0.05

Table 4. Computation times (in ms / block) of the three methods for three video sequences. The normalized complexity is calculated as the ratio between the average computation time for each search method and the reference time (3.60 ms/block) for translation-only motion estimation method.

Fig. 9 shows the empirical tradeoffs between the accuracies of these three search methods and their complexities. The best performance achievable is again observed in "Mobile Calendar" - an increase of 2.47 dB, 2.04 dB and 1.60 dB can be achieved with additional computational complexity of approximately 69%, 21% and 5% of that of the translation-only motion estimation.

5. Comparison with full affine transformation model

We want to compare the computational complexity of transformation estimation using Lie-operators to the complexity of transformation estimation using a full transformation model. We consider the affine model [8, 20, 21, 24], which was widely used in the literature to detect non-translation motions due to its ability to offer good compromise between complexity and performance. In its generic form, the 6-parameter affine model can be expressed as

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} a_1 x + a_2 y + a_3 \\ b_1 x + b_2 y + b_3 \end{pmatrix} \tag{39}$$

Since a_3 and b_3 in (39) are translational displacements, the 6-parameter affine model can be simplified to a 4-parameter model by estimating the translation motions using the conventional block matching method. In fact, even if the more complex gradient descent method is used for motion parameter estimation, to assure convergence, translation motion estimation is often employed as an initial stage that computes a coarse estimate of the translation component of the set of the motion parameters, so that the starting point of the gradient descent should be within the "basin" of the global minimum [3, 8].

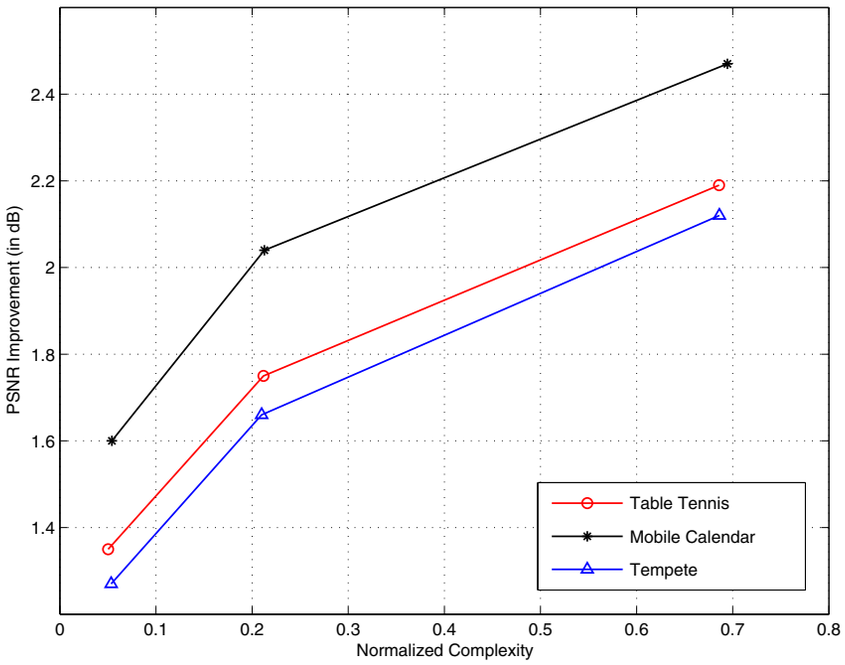

Figure 9. Increased accuracy vs. complexity. For each of the three sequences, from the right to the left, the three operating points correspond to the DP-like search, iterative search and the serial search methods, respectively. The normalized complexity is calculated as the ratio between the computation time for each search method and that for the translation-only motion estimation method as shown in Table 4.

5.1. A five-parameter affine transform model

Based on the above discussions, we choose the following affine motion model, which was used in [6] to improve the local translation motion compensation by taking into account rotation and scaling of small objects.

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} K_x & 0 \\ 0 & K_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{40}$$

The five parameters in (40) are estimated by using a two-step search method [6, 22]. First, parameters (t_x, t_y) corresponding to the translational motion between blocks in the current frame and the reference frame are searched for. This is a common step also shared by the Lie-operator approach (see Fig. 1), which operates on top of the match block yielded by the conventional translation block matching process. In the second step, the remaining three parameters for rotation and scaling (θ, K_x, K_y) are searched for. For ease of coding, θ, K_x and K_y are chosen from small sets of discrete values. For example, $\theta \in [-0.02\pi, 0, 0.02\pi]$, and $K_x, K_y \in [0.9, 1.0, 1.1]$ were chosen in [6]. On the other hand, the Lie-operator method is also suitable for the estimation of these small degrees of transformation. For example, the iterative approach discussed in Section 4.3 with three operators (R, S_x and S_y in Table 1) can be employed. Since (u,v) calculated by (40) can be real numbers, the pixel values at (u,v) have to be interpolated from the pixel values of the surrounding pixels. Bilinear interpolations are often employed [6][24, pp. 59]. More specifically, we assume that the four surrounding pixels in the reference frame have values $I_{\lfloor u \rfloor, \lfloor v \rfloor}$, $I_{\lfloor u+1 \rfloor, \lfloor v \rfloor}$, $I_{\lfloor u \rfloor, \lfloor v+1 \rfloor}$, and $I_{\lfloor u+1 \rfloor, \lfloor v+1 \rfloor}$, where $\lfloor s \rfloor$ is the *floor* function, which returns the nearest integer less than or equal to s. Thus the signal

value at (u,v) can be interpolated as

$$I_{u,v} = I_1 + r_1 \cdot (I_2 - I_1), \tag{41}$$

where

$$I_1 = I_{\lfloor u \rfloor, \lfloor v \rfloor} + r_2 \cdot \left(I_{\lfloor u+1 \rfloor, \lfloor v \rfloor} - I_{\lfloor u \rfloor, \lfloor v \rfloor} \right), \tag{42}$$

$$I_2 = I_{\lfloor u \rfloor, \lfloor v+1 \rfloor} + r_2 \cdot \left(I_{\lfloor u+1 \rfloor, \lfloor v+1 \rfloor} - I_{\lfloor u \rfloor, \lfloor v+1 \rfloor} \right), \tag{43}$$

and

$$r_1 = v - \lfloor v \rfloor, \quad r_2 = u - \lfloor u \rfloor. \tag{44}$$

Clearly, there will be extra computation cost incurred by these interpolation operations, which is not required by the Lie-operator approach.

5.2. Comparison of computational complexity

We now analyze the complexity required by motion estimation using the affine model described in Section 5.1, and the iterative Lie-operator approach described in Section 4.3, which can offer variable tradeoffs between the increased estimation accuracy and computational complexity by varying the number of iterations. The computational complexity is estimated by counting the number of additions/subtractions (C_{add}), and the number of multiplications (C_{mult}).

As shown in Table 5, the complexity of applying the affine model is

$$C_{Affine} = (C_{add}, C_{mult}) = (12MW^3, 12MW^3), \tag{45}$$

since one has to search for the best combination of the three types of motion parameters from W^3 possible choices, where W is the dimensionality of the candidate set for each motion parameter, which is assumed to be the same for each type of parameter, for ease of analysis and without much loss of generality. In the case of the above affine model given in Section 5.1, $W = 3$ was chosen [6].

Operation	C_{add}	C_{mult}
$u = xK_x \cos\theta + yK_y \sin\theta$, $v = -xK_x \sin\theta + yK_y \cos\theta$ (by Eq.(40))	$2M$	$8M$
Bilinear interpolation for $I(u,v)$ (by Eqs. (41)-(44))	$8M$	$3M$
MSE calculation (by Eq.(36))	$2M$	M
Total complexity / one combination of (θ,K_x, K_y)	$12M$	$12M$

Table 5. Number of arithmetic operations (per block) required by the transformation estimation using the affine model in (40), based on a displaced block with motion vector (t_x,t_y). Assume that values of $\sin\theta$ and $\cos\theta$ can be obtained by looking up from a pre-calculated table, and that M is the number of pixels in a block.

On the other hand, the complexity of the iterative Lie-operator approach is given in Table 6 for each iteration involving three operators (R, S_x and S_y). Therefore, if Q iterations are used, the total complexity is

$$C_{Lie} = (C_{add}, C_{mult}) = (M(5+9W)Q, M(8+6W)Q). \tag{46}$$

Type	Operation	C_{add}	C_{mult}
R	$\frac{\partial B}{\partial x}$ and $\frac{\partial B}{\partial y}$ (by Eq.(33))	$2M$	$2M$
	$L_R(B) = y\left(\frac{\partial B}{\partial x}\right) - x\left(\frac{\partial B}{\partial y}\right)$ (by Eq.(32)) calculated once for W distinct θ_R values	M	$2M$
	$B_R^{\theta_R} = B + \theta_R \times L_R(B)$ (by Eq.(34)) repeated for W distinct θ_R values	MW	MW
	MSE calculation (by Eq.(36)) repeated for W distinct θ_R values	$2MW$	MW
	Sub-total for R	$3M(1+W)$	$2M(2+W)$
S_x	$\frac{\partial B}{\partial x}$ (by Eq.(33))	M	M
	$L_{S_x}(B) = x\left(\frac{\partial B}{\partial x}\right)$ (see Table 1) calculated once for W distinct θ_{S_x} values	0	M
	$B_{S_x}^{\theta_{S_x}} = B + \theta_{S_x} \times L_{S_x}(B)$ (see Table 1) repeated for W distinct θ_{S_x} values	MW	MW
	MSE calculation (by Eq.(36)) repeated for W distinct θ_{S_x} values	$2MW$	MW
	Sub-total for S_x	$M(1+3W)$	$2M(1+W)$
S_y	Sub-total for S_y (same as that for S_x)	$M(1+3W)$	$2M(1+W)$
Total complexity of (R, S_x and S_y) / iteration		$M(5+9W)$	$M(8+6W)$

Table 6. Number of arithmetic operations required for finding the best transformed block (per iteration) using the iterative Lie-operator approach, based on a displaced block with motion vector (t_x,t_y). In each iteration, three types of operators (R, S_x, and S_y) are considered. Assume that M is the number of pixels in a block, and that the dimensionality of the set of candidate θ_R values for operator R is W, which is the same for the parameters θ_{S_x} and θ_{S_y} (for operators S_x and S_y, respectively).

From (45) and (46), it can be shown that as long as the number of candidate parameters for each type of motion $W \geq 3$, we have $C_{Lie} < 0.3\ C_{Affine}$ if the number of iterations $Q = 3$; and $C_{Lie} < 0.4\ C_{Affine}$ if $Q = 4$. The larger the W value is, the smaller the complexity of the iterative Lie operator becomes, relative to that of the estimation using the affine model.

Table 7 summarizes the increased accuracies obtained empirically for the iterative Lie-operator approach (using operators R, S_x and S_y in each iteration) and the affine model approach discussed in Section 5.1, which searches for the best combination of the parameters for rotation and scaling (θ, K_x, K_y), where $\theta \in [-0.02\pi, 0, 0.02\pi]$, and $K_x, K_y \in [0.9, 1.0, 1.1]$. The corresponding set of parameters for the Lie operators are thus chosen to be $\theta_R \in [-0.02\pi, 0, 0.02\pi]$, and $\theta_{S_x}, \theta_{S_y} \in [-0.1, 0, 0.1]$.

Search Method	Table Tennis			Mobile Calendar			Tempete		
	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg
Affine Model	1.58	0.24	1.14	1.63	0.99	1.41	1.29	0.85	1.14
RS_xS_y ($Q = 4$)	1.36	0.28	1.08	1.49	1.06	1.27	1.08	0.69	0.98
RS_xS_y ($Q = 3$)	1.23	0.22	0.95	1.35	0.94	1.11	0.93	0.60	0.84

Table 7. Increased estimation accuracy (in dB) of the iterative Lie-operator method versus that of the affine model approach. Q denotes the number of iterations.

It can be seen from Table 7 that with only 3 iterations, the Lie operator method performs closely to the affine model approach in terms of PSNR improvement; with one additional round of iteration, the Lie operator approach comes very close (within less than 0.1 dB) to the affine model approach. On a PC running Windows XP (with 3.40 GHz Pentium 4 CPU and 2GB RAM), the average running times of these two approaches were measured to be 0.46 ms/block (Lie operator, 4 iterations) and 1.46 ms/block (affine model). That is, Time (Lie operator, $Q = 4$) $\approx 1/3$ Time (affine model), which agrees with our analysis in Section 5.2. On the other hand, by comparing the data for iterative Lie operator approach in Table 3 and Table 7, it is obvious that the accuracy of the motion estimation can be increased significantly by using larger sets of candidate parameters (i.e., by increasing W in (46)) and considering more operators. Nevertheless, for the affine model, using a large W can lead to unacceptably large complexity, which increases linearly with W^3 in (45), as opposed to the almost linearly increased complexity of the Lie operator approach with W in (46). Therefore, the Lie operators have a clear advantage in terms of computational complexity, as long as they can provide good approximations to small degrees of transformation. Nevertheless, in the case of large degrees of transformations, the search method based on the full affine transformation model would be more accurate than the fast method based on Lie operator.

6. Conclusion

Lie operators are useful for efficient handwritten character recognition. Multiple operators can be combined to approximate small degrees of object transformations, such as scaling, rotations and deformations. In this chapter, we first explained in a tutorial fashion the underlying theory of Lie groups and Lie algebras. We then addressed the key problem of transformation estimation based on Lie operators, where exhaustive full search method is often impractical due to its prohibitively huge computational complexity. To illustrate the design of computationally efficient transformation estimation algorithms based on Lie operators, we selected the subject of motion and transformation estimation in video coding as an example. We presented several fast search algorithms (including the dynamic programming like, serial, and iterative search methods), which integrated multiple Lie operators to detect smaller degrees of transformation in video scenes. We provided a detailed analysis of the varying tradeoffs between estimation accuracies and computational complexities for these transformation estimation algorithms. We demonstrated that non-translational transformation estimation based on Lie operators could be used to improve the overall accuracy of motion estimation in video coding, with only a modest increase of its overall computational complexity. In particular, we showed that the iterative search method based on Lie operators has much lower complexity than the transformation estimation method based on the full affine transformation model, with only negligibly small degradation in the estimation accuracy.

Author details

W. David Pan
Department of Electrical and Computer Engineering, University of Alabama in Huntsville, Huntsville, Alabama 35899, USA.

7. References

[1] B. Carpentieri, Block matching displacement estimation: a sliding window approach, Information Sciences 135 (1-2), (2001) 71–86.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, Second Edition, MIT Press, 2001.

[3] F. Dufaux, J. Konrad, Efficient, robust, and fast global motion estimation for videocoding, IEEE Trans. Image Processing 9 (3) (2000) 497–501.

[4] R. Gilmore, Lie Groups, Lie Algebras and Some of Their Applications, John Wiley & Sons, 1974.

[5] R. Gilmore, Lie Groups, Physics, and Geometry: An Introduction for Physicists, Engineers and Chemists, Cambridge University Press, 2008.

[6] H. Jozawa, K. Kamikura, A. Sagata, H. Kotera, H. Watanabe, Two-stage motion compensation using adaptive global MC and local affine MC, IEEE Trans. Circuits and Systems for Video Technology 7 (1) (1997) 75–85.

[7] T. C. T. Kuo, A. L. P. Chen, A mask matching approach for video segmentation on compressed data, Information Sciences 141 (1-2) (2002) 169–191.

[8] W. Li, J.-R. Ohm, M. van der Schaar, H. Jiang, S. Li, MPEG-4 video verification model version 18.0, in: ISO/IEC JTC1/SC29/WG11 N3908, Pisa, Italy, 2001.

[9] S. Lin, D. J. Costello, Error Control Coding, Second Edition, Pearson Prentice Hall, 2004.

[10] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, D. J. LeGall, MPEG Video Compression Standard, Chapman & Hall, 1996.

[11] M. Nalasani, W. D. Pan, On the complexity and accuracy of the lie operator based motion estimation, in: Proc. IEEE Southeastern Symposium on System Theory (SSST), Atlanta, Georgia, 2004, pp. 16–20.

[12] W. D. Pan, S.-M. Yoo, M. Nalasani, P. G. Cox, Efficient local transformation estimation using Lie operators, Information Sciences, 177 (2007) 815-831.

[13] W. D. Pan, S.-M. Yoo, C.-H. Park, Complexity accuracy tradeoffs of Lie operators in motion estimation, Pattern Recognition Letters 28 (2007) 778–787.

[14] C. A. Papadopoulos, T. G. Clarkson, Motion estimation using second-order geometric transformations, IEEE Trans. Circuits and Systems on Video Technology 5 (4) (1995) 319–331.

[15] H. Richter, A. Smolic, B. Stabernack, E. Muller, Real time global motion estimation for an MPEG-4 video encoder, in: Proc. Picture Coding Symposium (PCS), Seoul, Korea, 2001.

[16] H. Samelson, Notes on Lie Algebras, Springer, 1990.

[17] K. Sayood, Introduction to Data Compression, Morgan Kaufmann, 2000.

[18] P. Y. Simard, Y. A. LeCun, J. S. Denker, B. Victorri, Transformation invariance in pattern recognition - tangent distance and tangent propagation, International Journal of Imaging Systems & Technology 11 (3) (1998) 239–274.

[19] K. Sookhanaphibarn, C. Lursinsap, A new feature extractor invariant to intensity, rotation, and scaling of color images, Information Sciences 176 (14) (2006) 2097–2119.

[20] C. Stiller, J. Konrad, Estimating motion in image sequences, IEEE Signal Processing Magazine (1999) 70–91.

[21] Y. Su, M.-T. Sun, V. Hsu, Global motion estimation from coarsely sampled motion vector field and the applications, IEEE Trans. Circuits and Systems on Video Technology 15 (2) (2005) 232–242.

[22] Y. T. Tse, R. L. Baker, Global zoom/pan estimation and compensation for video compression, in: Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Toronto, Canada, 1991, pp. 2725–2728.
[23] Y. Wang, J. Ostermann, Y.-Q. Zhang, Video Processing and Communications, Prentice Hall, 2002.
[24] G. Wolberg, Digital Image Warping, IEEE Computer Society Press, 1990.

IntechOpen