Efficient local transformation estimation using Lie operators

W. David Pan a, Seong-Moo Yoo a,*, Mahesh Nalasani a, Paul G. Cox b

a Department of Electrical and Computer Engineering, University of Alabama in Huntsville, 301 Sparkman Dr, Huntsville, AL 35899, USA
b PERL Research, 3058 Leeman Ferry Rd., Huntsville, AL 35801, USA

Received 10 May 2005; received in revised form 17 June 2006; accepted 5 July 2006

Abstract

Conventional translation-only motion estimation algorithms cannot cope with transformations of objects such as scaling, rotations and deformations. Motion models characterizing non-translation motions are thus beneficial as they offer more accurate motion estimation and compensation. In this paper, we introduce low-complexity transformation estimation methods with four motion models based on Lie operators, which are linear operators that have found applications in optical character recognitions. We show that individual Lie operators are capable of capturing small degrees of object transformations. We propose an efficient local transformation estimation algorithm in order to further improve the accuracy of the translation-only estimation by integrating all four motion models. Simulations with an MPEG-2 video codec on two video sequences show that the proposed transformation estimation approach can noticeably improve the motion compensation performance of the translation-only method by achieving higher PSNR (peak signal-to-noise ratio) values for the predicted frames, with only a small fraction of the complexity required by the translation motion search.

© 2006 Elsevier Inc. All rights reserved.

Keywords: Motion estimation; Motion compensation; MPEG-2; Motion model; Lie operators

1. Introduction

Motion estimation is a critical component of almost every video coding system [20,29,39]. Most compression techniques exploit the temporal redundancy that exists between a frame. 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. This is partly because a translation model can be readily characterized by displacement motion vectors and thus has lower implementation complexity than other
motion models such as the affine and perspective models [34]. In this section, we first provide a brief introduction to the block-based translational motion estimation method that is widely used in many international standards. Next, we conduct a literature survey on both the global and local motion estimation methods that use non-translational models to improve the performance of motion estimation achieved by using the translation-only model. We then point out the major characteristics of the block-based transformation estimation method based on Lie operators, which is introduced in this paper to improve the performance of block-based, translation-only motion estimation method with low computational complexity.

1.1. Translation-only motion estimation

Block-based motion estimation [20,39] has been adopted in international standards for video coding such as MPEGs and H.263, 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 [2,16]. 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. For instance, in Fig. 1, the difference between the ball in the current frame and that in the previous frame would be very small, provided the ball can be captured accurately by the motion estimation. Fig. 2 shows two frames from another sequence “Mobile Calendar” used in the simulations.

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 based on mean square error (MSE). The block with the minimum MSE is considered to be the best match, and its associated motion vector (\(d_x, d_y\)) is given by

![Fig. 1. The 10th frame (left) and the 11th frame (right) from the video sequence “Table Tennis” are highly correlated: most regions on the background remain unchanged over these two frames. Note the deformation of the ball, which corresponds to part of the local motions undergone by the ball.](image1)

![Fig. 2. The 10th frame (left) and the 100th frame (right) from the video sequence “Mobile Calendar”. The scene is a combination of global motions (e.g., those due to the camera zooming out) and local motions including the ball moving and rotating in the foreground.](image2)
(dx, dy) = \arg \min_{(du, dv) \in [-R, R]} \left\{ \text{MSE}_{m,n} = \sum_{i,j=0}^{R-1} [F_2(x, y) - F_1(x + du, y + dv)]^2 \right\}, \quad (1)

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 $P_1$ is calculated as

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

where $\text{MSE}_{\text{avg}}$ is the average mean square error between $F_2$ and $P_1$ given by

$$\text{MSE}_{\text{avg}} = \frac{1}{M \times N} \sum_{m=0}^{M} \sum_{n=0}^{N} \text{MSE}_{m,n},$$

where $\text{MSE}_{m,n}$ is defined in (1), 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. For example, in Fig. 1, the small region surrounding the ball in the current frame appears to be a deformed and scaled-up version of the ball in the previous frame. This additional information about the correlation between neighboring frames can be exploited by considering non-translational motion models to increase the accuracy of the motion estimation.

### 1.2. Motion models

There have been numerous previous works on the use of non-translational motion models. For example, Wu and Kittler [43] proposed a differential method (using Taylor series expansion) for simultaneous estimation of rotation, change of scale and translation, where four unknown parameters used to specify the coefficients of the mapping functions are estimated for each $48 \times 48$ pixels block. Better prediction in subjective form was reported than the two parameter translation-only model, however, no quantitative results were given. Hoetter [9], like Wu and Kittler, used a second-order Taylor series expansion of the image signal to compensate only for the global zoom and pan of the image caused by camera. Keesman [14] used a six parameter motion model called “slanting model” to compensate for translation, rotation, zooming, and shearing. An improvement of about 3 dB was reported in the prediction PSNR of $21 \times 21$ blocks as compared to the conventional translation-only motion model. Moreover, in [10,21], a motion compensation algorithm based on an eight-parameter perspective model was introduced to describe arbitrary 3D motion of a planar rigid object. Papadopoulos and Clarkson [25] extended Keesman’s affine transform motion model into second-order geometric transformations (GT) that employ 12 parameters capable of describing uneven shearing. Their GT-based motion compensation scheme can provide over 3 dB gains on $16 \times 16$ blocks. For a thorough exposition of the subject of motion models and their applications in motion estimation, refer to [34,42].

### 1.3. Global vs. local motion estimation

In general, motion in a sequence of images results from motion of the camera (e.g., panning, zooming, tilting and/or a more complex combination of these basic components), as well as from the translational displacements and transformations (e.g., scaling, rotations, and deformations) of individual objects composing the
scene. The former is often referred to as global motion where the region of support for motion representation consists of the entire image frame [4], and the latter as local motion [3], where the region of support may consist of small regions (e.g., rectangular blocks or even a single pixel) within an image [5,34].

Estimating global and local motion separately not only results in more precise estimates, but it also leads to a more compact representation of the motion information by using a few model parameters. Consequently, tools based on global motion estimation (GME) have been adopted by the MPEG-4 standard, including the static-sprite [44] generations in MPEG-4 Sprite coding [32], and global motion compensation in MPEG-4 advanced simple profile [11,17]. Techniques for global motion estimation (GME) can be categorized roughly into direct and indirect methods. Direct GME methods, such as those based on the estimation of the perspective model parameters in [3,5,17], attempt to minimize the prediction errors in the pixel domain. Direct methods tend to be computationally expensive due to the iterative processes in the non-linear estimation and the number of pixels involved. [15] presents a fast GME method suitable for both affine and perspective models, which involves only a small subset of the image pixels in the estimation process. On the other hand, indirect GME methods typically contain two stages, with GME performing at the second stage based on the motion vectors resulting from the first motion estimation stage [8,12,28,31,35,37,38]. A study was conducted in [27] comparing the direct and indirect GME methods in terms of their estimation accuracy, computational complexity and coding efficiency.

Although the GME methods proposed in [1,5,15,35] are capable of significantly reducing the residual of motion compensation and thus the entropy of the local motion fields, they cannot simply replace the local motion estimation. For example, in Fig. 2, a GME method using an affine or perspective model could be employed to capture the global motions existing in the “Mobile Calendar” sequence due to the camera zooming, however, local motions (e.g., rotation) of the ball in the foreground could not be accounted for properly by a global method. In fact, the main difficulty in estimating global motion parameters lies in the existence of independently moving objects that may create outliers and thus introduce a bias to the estimated parameters [38]. To remove the influence of such outliers, a robust estimator technique called the M-estimator (maximum likelihood type estimator) [26] was used in [5,28,31] to limit the sensitivity of the estimation process to outliers. The robustness of the M-estimator could be improved by means of motion vector field segmentation as a preprocessing step. In [35], an outlier-rejection method based on histogram thresholding is employed to alleviate outlier-induced deviations of the global motion models estimated from coarsely sampled motion vector fields.

Therefore, even if global motion estimation using motion models is effective in improving the accuracy of the motion estimation and compensation, more accurate local motion estimation is still highly desirable, which is the focus of this paper. A two-stage motion compensation procedure was proposed in [37]. In the first stage, a global motion estimator is used to construct a globally compensated frame, which is used by the succeeding local motion compensation in the second stage to model the remaining object motion. A similar two-stage approach is also found in [12]. However, rather than using the translation-only motion estimator in the second stage for macro-block prediction, an affine model is applied locally to capture complex object motions such as rotation, scaling and deformation. More sophisticated motion models (such as the 6-parameter [14] and 12-parameter [25] models mentioned in Section 1.2) can be employed to further improve the accuracy of block-based local motion estimation, albeit at the expense of much increased computational complexity. Clearly, better performance is expected to be achieved by applying motion models on individual objects [10,21] instead of on evenly-partitioned blocks. Nonetheless, the block-based motion estimation approach is widely adopted by many video coding standards, mainly due to its regularity, and low complexity in both software and hardware implementations. Thus, if standard-compatible, block-based local motion estimation is desired, then a wish list for the ideal method can be compiled below:

- It is capable of achieving good performance by using motion models to account for those non-translation motions experienced by individual objects.
- The number of parameters in the chosen motion model should be limited to avoid “overmodeling” that may actually decrease the motion estimation accuracy [34]. In addition, since extra bits are required to code these parameters, an excessive number of parameters results in increased coding cost that adversely offsets the coding gains brought by improved motion estimation accuracy.
• The estimation of the parameters motion model should have low complexity, since estimation has to be repeated for all the blocks, rather than merely once as with the case of GME.
• The number of blocks where motion models are estimated should also be limited. Therefore, local motion estimation that can be applied adaptively on selected blocks will lead to savings in both complexity and bit rates.

We can see from the above list that some of the goals conflict with each other, e.g., the demand for excellent performance in accuracy and the need to lower the complexity of motion estimation. The key is to strike a good balance in view of the tradeoffs between conflicting factors. In this paper, we introduce low-complexity, linear motion models supported by the Lie operators, which can effectively detect and compensate small degrees of rotation, scaling and deformations of objects in the scene. Our goal is to improve the accuracy of the conventional block-based local motion estimator using the translation-only motion model.

1.4. Efficient Lie-derivative method for local motion estimation

We introduce two methods that are based on the Lie operators derived from Lie derivatives. The first one is designed for transformation estimation with an individual motion model. While Gerken and Adolph [6] adopted a similar approach in investigating the additional gain attainable by using rotational motion compensation, our second approach is an iterative algorithm for transformation estimation with integrated multiple motion models (more than the rotations), which allow for more accurate prediction. Both methods are block-based and thus they are compatible with the framework of existing international standards such as MPEGs and H.26x, which support only translation motion estimation. Furthermore, both methods can be used as a “plug-in” component that can be easily applied on top of any translation-only motion estimation scheme for its refinement. More importantly, due to its inherent constraint of small degree transformations, these Lie-operator based methods incur much lower computational complexity than the computationally expensive parameter estimation associated with other motion models such as the affine and perspective models when used in local motion estimation [12,14,25,43]. A detailed comparison between the motion models supported by the Lie-operators and other well-known models such as the affine and perspective models is given in Section 2.2. An analysis of the computational complexity of the Lie-operator approach is provided in Section 3.4.

Simulations are conducted on two test sequences. Results demonstrate that remarkable improvements (up to about 3.9 dB and about 1.4 dB on average) in estimation accuracy can be achieved by applying the transformation estimation approach with integrated motion models. Furthermore, we evaluate the performance of our transformation estimation algorithms when they are embedded in an MPEG-2 codec, where prediction errors tend to propagate to succeeding frames. The results show that transformation estimation using Lie operators can lead to additional gains (about 1.4 dB) for predicted MPEG-2 frames in two standard test sequences.

The remainder of the paper is organized as follows. Section 2 introduces the Lie operators and the supported motion models, which are compared against some existing parametric motion models. The two transformation estimation algorithms based on Lie operators are then presented in Section 3, including an analysis of their computational complexities. In Section 4, the performance of the proposed transformation estimation methods are evaluated in the context of an MPEG-2 codec. Finally, conclusions are drawn in Section 5.

2. Introduction to Lie operators

2.1. Motion models and their associated Lie operators

Lie derivatives were introduced in [30] as an effective method for handling transformation invariance [33] in pattern recognition. Lie operators derived from Lie derivatives are operators in the Lie Algebra with low computational complexity. In this paper, four motion models based on the Lie algebra are adopted from [30]: rotation ($R$), scaling ($S$), parallel deformation ($P$), and diagonal deformation ($D$). We use rotation as an example to illustrate the basic idea of these motion models. A pixel with the coordinate $(x, y)^T$ will be rotated with a degree $\theta$ to a new position with the following coordinate $(u, v)^T$: 

\[
(u, v)^T = (x, y)^T R(\theta) 
\]

where $R(\theta)$ is the rotation matrix.
Table 1

Four motion models and their associated Lie operators ($\theta$ is the degree of the transformations)

<table>
<thead>
<tr>
<th>Motion model</th>
<th>Transformation matrix $T_\theta$ (adapted from [30])</th>
<th>Lie operator and the transformed image</th>
</tr>
</thead>
</table>
| $R$          | \[
\begin{pmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{pmatrix}
\] | $L_R = y \frac{\partial}{\partial x} - x \frac{\partial}{\partial y}$, $I_R = I + \theta \times L_R(I)$ |
| $S$          | \[
\begin{pmatrix}
1 + \theta, & 0 \\
0, & 1 + \theta
\end{pmatrix}
\] | $L_S = x \frac{\partial}{\partial x} + y \frac{\partial}{\partial y}$, $I_S = I + \theta \times L_S(I)$ |
| $P$          | \[
\begin{pmatrix}
1 + \theta, & 0 \\
0, & 1 - \theta
\end{pmatrix}
\] | $L_P = x \frac{\partial}{\partial x} - y \frac{\partial}{\partial y}$, $I_P = I + \theta \times L_P(I)$ |
| $D$          | \[
\begin{pmatrix}
1, & \theta \\
\theta, & 1
\end{pmatrix}
\] | $L_D = y \frac{\partial}{\partial x} + x \frac{\partial}{\partial y}$, $I_D = I + \theta \times L_D(I)$ |

$t_\theta = \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} u \\ v \end{pmatrix} = T_\theta \begin{pmatrix} x \\ y \end{pmatrix},$ \hspace{1cm} (4)

where $t_\theta$ is the transformation (which is rotation in this case) and $T_\theta$ is the transformation matrix given by

$T_\theta = \begin{pmatrix}
\cos \theta, & -\sin \theta \\
\sin \theta, & \cos \theta
\end{pmatrix}.$ \hspace{1cm} (5)

Based on the above rotational motion model characterized by its transformation matrix, a corresponding Lie operator $L_R$ can be derived via differentiation (derivative taking) (see [30] for details). The definitions of the other three motion models ($S$, $P$ and $D$) and their associated Lie operators ($L_S$, $L_P$ and $L_D$) are summarized in Table 1.

By applying the $R$ (rotation) operator on the original image (or block) $I$, an rotated image (or block) $I_R$ can then be readily approximated. As long as the degree of rotation $\theta$ is very small, we can have good approximation.

In order to apply a Lie operator, which involves derivatives, we first convert a discrete image frame $I$ into a continuous one $f$ by means of convolution: $f = I \ast g_\sigma$, where $g_\sigma$ is a 2D Gaussian function defined as [30]:

$g_\sigma = \exp \left(-\frac{x^2 + y^2}{2\sigma^2}\right).$ \hspace{1cm} (6)

In the case of rotation, after applying the operator $L_R$, we have

$L_R(f) = \left( y \frac{\partial}{\partial x} - x \frac{\partial}{\partial y} \right) (I \ast g_\sigma) = y \left( I \ast \frac{\partial g_\sigma}{\partial x} \right) - x \left( I \ast \frac{\partial g_\sigma}{\partial y} \right).$ \hspace{1cm} (7)

To avoid the high computational complexity associated with the convolution operation and the calculation of the partial derivatives of the Gaussian function in (6), we can apply the Lie operator on the discrete image directly, by using the following approximations [25]

$L_R(f) \approx L_R(I) = \left( y \frac{\partial}{\partial x} - x \frac{\partial}{\partial y} \right) I = y \left( \frac{\partial I}{\partial x} \right) - x \left( \frac{\partial I}{\partial y} \right),$ \hspace{1cm} (8)

where $\frac{\partial I}{\partial x} \approx \frac{1}{2} [I(x+1,y) - I(x-1,y)],$ and $\frac{\partial I}{\partial y} \approx \frac{1}{2} [I(x,y+1) - I(x,y-1)].$ After the Lie operator is applied, the rotated version of the frame $I$ can then be easily obtained as $I_R = I + \theta \times L_R(I)$. Similarly, we can derive the transformed frames for other motion models, which are summarized in the third column of Table 1.

We can see from (8) 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 ($\theta$) can be obtained by using the same $L_R(I)$. Therefore, the implementation of Lie operators has fairly low computational complexity. A detailed analysis of the computational complexity of the Lie-operator approach is given in Section 3.4.
Table 2

Various parametric motion models

<table>
<thead>
<tr>
<th>Model</th>
<th>Number of parameters</th>
<th>Transform ((x,y) \mapsto (u,v))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometric [43,35]</td>
<td>4</td>
<td>(u = a_1x - a_2y + a_3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(v = a_4x + a_5y + a_6)</td>
</tr>
<tr>
<td>Affine [34,42]</td>
<td>6</td>
<td>(u = a_1x + a_2y + a_3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(v = b_1x + b_2y + b_3)</td>
</tr>
<tr>
<td>Perspective [42,5,35]</td>
<td>8</td>
<td>(u = a_1x + a_2y + a_3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(v = \frac{b_1x + b_2y + b_3}{c_1x + c_2y + 1})</td>
</tr>
<tr>
<td>Second-order polynomial [25,40,42]</td>
<td>12</td>
<td>(u = a_1x^2 + a_2xy + a_3y^2 + a_4x + a_5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(v = b_1x^2 + b_2xy + b_3y^2 + b_4y + b_5)</td>
</tr>
</tbody>
</table>

2.2. Comparisons with existing motion models

It is clear in Table 1 that \(\theta\) is the only parameter for each motion model \((R, S, P\) and \(D\)), which will significantly simplify the parameter estimation procedure due to the degree of freedom being 1. This advantage is absent in several other motion models in the literature, which are listed in Table 2. Clearly, all models in Table 2 have degrees of freedom much greater than 1, thereby in general requiring much higher complexity in estimation parameters than those models in Table 1.

Since the Lie-operator based motion models are applied on the translation-compensated frames, parameters (e.g., \(a_3\) and \(a_4\) for the geometric model, and \(a_5\) and \(b_3\) in the affine and perspective model in Table 2) describing translation need not be included in the model. As pointed out in Section 1, low-order models tend to be more suitable (than the perspective and second-order polynomial models) for local motion estimation since they are not too complex and thus offer a good compromise between flexibility and applicability. In Table 2, the 4-parameter geometric model is a special case of the 6-parameter affine model. Likewise, the one-parameter models in Table 1 can be viewed as special cases of the affine model. For example, the scaling model \(S\) is equivalent to an affine model with \(a_1 = b_2 = 1 + \theta\), and \(a_2 = a_3 = b_1 = b_3 = 0\). The 4-parameter geometric model in Table 2 is fairly restrictive in that not only it cannot describe sheer motion, but also it assumes that transformations are applied in the strict order of rotation, then scaling and finally translation [43]. Due to the non-communicative property of different kinds of motion, we propose an algorithm in Section 3.3 that can further improve the accuracy of motion estimation by searching for a good order of applying the four motion models (listed in Table 1) sequentially.

3. Motion estimation using lie operators

Initially, the four Lie operators \((R, S, P\) and \(D\)) introduced in Section 2.1 are performed individually on each block of the predicted frame \(P_1\), to determine the improvement in PSNR. Later, we introduce an efficient algorithm that can combine all operations to achieve the best possible PSNR improvement in a computationally efficient way.

3.1. Transformation estimation using individual motion models

As illustrated in Fig. 3, 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. 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 \(\theta\) 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 $\text{PSNR}_2$. 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 current frame and the predicted frame. The improved accuracy due to the motion models is calculated as $(\text{PSNR}_2 - \text{PSNR}_1)$. The improvement in the accuracy of motion estimation can be further increased by integrating motion models ($R$, $S$, $P$, and $D$). In the next subsection, we propose an algorithm that can give better PSNR improvements by integrating the four Lie operators.

### 3.2. Significance of $\theta$

One of the most critical parameters in the transformation estimation is $\theta$, as it controls the amount of transformation on each motion block. In actual implementations, transformation estimation is applied on the block-by-block basis, where the block size is chosen to be very small (e.g., $4 \times 4$ blocks) to account for small objects. In transformation estimation, we search for $\theta$ that best matches the degree of transformation the object has experienced between the current frame and the previous frame. If $\theta$ is found to be 0 in the case of rotation, then the resultant rotated block will be the same as the input block. If $\theta$ has a larger search range,
then the probability of finding a better $\theta$ may be increased, however, the complexity of searching will be increased as well, since we have to examine more candidates.

On the other hand, the step size of $\theta$ 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 $\theta$ without increasing the search complexity. Extensive experiments have been conducted to determine a suitable search range of the $\theta$ values by examining the histograms of $\theta$. The step sizes can also be determined empirically. Fig. 4 shows the histogram of $\theta$ for 300 frames (with block size of $4 \times 4$) of the “Table Tennis” sequence in Fig. 1 for all four models ($R$, $S$, $P$, and $D$). We can see that the frequency (probability) of $\theta$ is approaching zero near the tails ($-0.14$ and $0.14$) of the histogram. Thus, it is feasible to significantly reduce the total number of candidate parameters by focusing the search on those transformations with smaller degrees (close to zero). In the simulations, the range of $[-0.14, 0.14]$ with a fixed step size of $\theta (=0.02)$ is chosen for $\theta$. 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 $\theta$ values, whereas the step size increases as the search drifts away towards the tails of the range of the candidate $\theta$ values [22].

3.3. Transformation estimation with integrated motion models

The method described in Section 3.1 is restrictive in that we can apply only one of the four operators at a time. It would be desirable to choose the most suitable operator adaptively in order to maximize the prediction performance. For example, rotation may be the best operator if there are more rotational objects in the scene;

![Flowchart](image)

Fig. 5. The flowchart for the motion estimation method with integrated motion models of $R$ (rotation), $S$ (scaling), $P$ (parallel deformation) and $D$ (diagonal deformation). Each block $B_T[I]$ (where $1 \leq I \leq I_{max}$, and $I_{max}$ is the total number of blocks in a frame) of the predicted frame $P$ is transformed by applying the best integrated motion model into $B_T[I]$, from which a more accurate predicted frame $P_T$ is formed.
and scaling may be the best operator to capture the object transformation due to excessive camera zooming. Sometimes both of these operators are effective in describing the transformations when a similar amount of zooming and rotation have occurred. We therefore propose an integrated algorithm (shown in Fig. 5) to combine these operators and estimate the best combinations of these operators on each block of frame \( P \) to improve the estimation accuracy.

The accuracy of the motion estimation can be improved by considering all four motion models for each frame, i.e., for each block, the best one among four operators will be applied. The operator with the smallest MSE is chosen as the best operator for each block. This can capture all transformations of objects in one particular frame such as scaling, rotations and deformations. In fact, we can further increase the prediction accuracy by applying the integrated motion models iteratively. In Fig. 5, the transformed block \( B_T^I \) can be fed back to the input of the transformation estimation module so that it will be further transformed to achieve an even better predicted block than \( B_T^I \). The iteration can be terminated after no significant improvement of the prediction accuracy is observed, or after a preset number of iterations.

### 3.4. Complexity analysis

We start with analyzing the complexity required by the transformation estimation using individual Lie operators. We measure the computational complexity by counting the number of two major arithmetic operations, as denoted by the pair (number of multiplications, number of additions). Let us consider the rotation operator as an example. Following the discussions in Section 2.1, the complexity of the major steps leading to a transformed block is listed in Table 3, which shows that the total complexity of applying the \( R \) operator on block \( B \) is \((4M, 3M)\).\(^1\) Note here again that the resulting block \( LR(B) \) has to be computed just once.

Once \( LR(B) \) is available, for any given candidate \( \theta \) value, it takes additional \((M, M)\) operations to calculate the transformed block \( B_R^\theta \) (with degree being \( \theta \)). If the MSE (mean square error) is the search criteria, then it takes \((M, 2M)\) operations to determine the MSE between the transformed block \( B_R^\theta \) and the block to be predicted in the current frame. Hence a total of \((2M, 3M)\) operations are needed for any given candidate \( \theta \) value.

If the total number of candidate \( \theta \) values is \( S \), then the overall complexity required for the transformation estimation is

\[
C_{\text{Lie}} = ([2(S + 2)M], [3(S + 1)M]).
\]

Note the cost for the comparison operation is ignored here.

For comparison, the complexity of the conventional translation-only motion search is estimated to be

\[
C_{\text{Trans}} = (RM, 2RM),
\]

where \( R \) is the number of candidate blocks considered.

---

\(^1\) See Eq. (8). Here we replace \( I \) by \( B \) to emphasize the fact that the Lie operator is applied on a block \( B \), instead of an entire image \( I \).
It follows from (9) and (10) that as long as
\[ S \leq 0.5R - 2, \]  
we have \( C_{\text{Lie}} \leq C_{\text{Trans}} \) (true for both the multiplication and addition components). Otherwise, we have \( C_{\text{Lie}} > C_{\text{Trans}} \). Note that the block size \( M \) does not appear in the condition given in (11) for the transformation estimation based on Lie operators to be faster than the translation motion search. As mentioned in Section 3.2, if the range of \([-0.14, 0.14]\) with a fixed step size of 0.02 is chosen for \( \theta \), then \( S = 14 \).\(^2\) Let us consider two feasible methods of performing translation motion search (assuming that the search window is chosen to be \pm 15 \text{ pixels}).

- Exhaustive full search. Therefore, \( R = (2 \times 15 + 1)^2 = 961 \). Obviously, \( S = 14 \ll (0.5R - 2) = 478.5 \). Therefore, \( C_{\text{Lie}} \ll C_{\text{Trans}} \).
- Fast search using the logarithmic method [39]. Given the search window size, four-step search can be employed, with step sizes being 15, 8, 4, and 2. At each step, eight candidate blocks will be searched. Hence there are a total of \( R = 32 \) candidate blocks. In this case, \( S = 0.5R - 2 = 14 \), we thus expect that the transformation estimation method based on individual Lie-operator has roughly the same complexity as this fast translation search method.

We have shown that the \( R \) model has low computation complexity. From the third column of Table 1, it can be easily seen that parameter estimation with the other three motion models has the same complexity as that of the \( R \) model. Therefore, the complexity of the algorithm integrating all four models (described in Section 3.3) is approximately
\[ C_{\text{integ}} = 4Q \times C_{\text{Lie}}, \]  
where \( Q \) is the number of iterations used. It follows from (11) that as long as
\[ S < \frac{R}{8Q} - 2, \]  
we have \( C_{\text{integ}} < C_{\text{Trans}} \). If \( Q = 4 \), \( S = 14 \), and \( R = 961 \) (as in the case of full search), then we can expect that the relation \( C_{\text{integ}} \ll C_{\text{Trans}} \) still holds.

If fact, experimental results in [22] show that \( S \) can be reduced (e.g., to 7) by using non-uniform step sizes for \( \theta \), while the accuracy of transformation estimation could still be preserved. Consequently, the complexity of the Lie-operator based method can be further lowered.

### 3.5. Issues in practical implementations

In practice, extra bits have to be assigned to code \( \theta \) at the encoder and are sent to the decoder as side information. If the number of candidate \( \theta \) values is \( S \), then \( \lceil \log_2 S \rceil \) bits are required for each block. Decreasing \( S \) serves to not only reduce the search complexity significantly, but also to reduce the bit rates for the side information. For example, reducing \( S \) from 15 to 7 can reduce 1 bit/block for coding \( \theta \), which may amount to 4 bits/block in the case of integrated motion models to be discussed in Section 3.3 (with four iterations).

Since the total complexity and the number of bits for coding \( \theta \) increase linearly with the number of blocks on which the local transformation estimation is applied, the Lie operators should be used selectively on those blocks in the regions where noticeable changes can be detected over time. These moving regions can be detected by, for example, finding the difference between the neighboring frames, followed by the thresholding operation [7,24]. Therefore, if the background of the scene does not change significantly, then transformation based on Lie operators will not be performed on those blocks that belong to the background. On the other hand, even if the uncovered background may be classified as a “moving” region, transformation estimation will not likely find any good match for the objects in the uncovered background. Furthermore, light condition may also have an impact on the objects in the scene. If an object happens to be scaled or deformed to some extent due to changes in illumination, then the Lie operators are supposed to be able to capture this type of

\(^2\) It is not necessary to compare with the block with \( \theta = 0 \), which is indeed the block to be transformed.
special transformations of the object. In this paper, for the sake of uniformity, we apply the Lie operators to all the blocks in a frame. Therefore, the resulting computational complexity and the amount of side information for coding the $\theta$ values can be viewed as an upper bound, since Lie operators can be applied on only a small portion of the blocks in the “moving” regions. Other techniques such as the merge-and-split method in [18] can be used to optimally balance the reduced bit rates in coding motion-compensation errors (due to more accurate motion estimation by using motion models), and the increased bit rates in coding motion information (due to the extra side information).

On the other hand, the number of blocks with transformation estimation can be reduced by using larger blocks (e.g., $16 \times 16$, or even $32 \times 32$ blocks). However, a large block may contain multiple objects with different motions. Consequently, the accuracy of the motion estimation may be sacrificed. Therefore, the choice of the block size in motion estimation is often dictated by the tradeoff between bit rates and the accuracy of motion estimation. In MPEG-2 [20], motion estimation is performed on the basis $8 \times 8$ blocks. To increase the accuracy of motion estimation, the H.264/AVC video coding standard allows more flexible and efficient selection of block sizes and shapes than MPEG-2, with a minimum motion compensation block size as small as $4 \times 4$ [23,36,41]. The need for extra bits for coding the additional motion vectors associated with smaller blocks is justified by the significant improvement achieved by H.264/AVC – a 50% coding gain over MPEG-2 is reported in [13]. While it may be more suitable to use $8 \times 8$ blocks for the video sequence that contains a great number of relatively large objects, the accuracy of the transformation estimation using Lie operators may degrade, due to the underlying assumption of single object being violated. Hence, in the spirit of H.264/AVC, a small block size of $4 \times 4$ is chosen in the simulations in Section 4 to ensure the locality of the transformation estimation. In actual implementations, the side information associated with coding transformation parameters can be reduced substantially by means of a host of existing techniques. For example, analogous to the adaptive approach adopted by H.264/AVC, after the residual error associated with transformation estimation using Lie operators is obtained for each of the four $4 \times 4$ blocks within an $8 \times 8$ MPEG-2 block, we can choose either one of the following three modes of motion compensation for the $8 \times 8$ block: (i) each of the $4 \times 4$ blocks will be motion-compensated individually, given that their residual errors are sufficiently small and they have distinct transformation parameters. This mode is chosen because the more accurate motion estimation justifies the extra side bits required; (ii) all four $4 \times 4$ will be merged into the original $8 \times 8$ block, which will be motion-compensated using one identical set of transformation parameters. This mode is selected because these $4 \times 4$ blocks actually belong to one object, and thus they undergo similar amount of transformation; (iii) if the transformation estimation was unable to bring much improvement in the accuracy of motion estimation, then we will fall back to the conventional mode of translation-only motion compensation for the $8 \times 8$ block in MPEG-2. To further reduce the side information, other methods can also be considered, e.g., we can set the number of candidate $\theta$ values in the transformation estimation, and use the sophisticated variable-length coders for coding the transformation parameters, the treatment of which is beyond the scope of this paper.

3.6. Simulation results

Standard test sequences “Table tennis” [19] and “Mobile calendar” are used in our simulations. Both sequences contain 300 frames of images in the CIF format ($288 \times 352$ pixels). Block size of $4 \times 4$, searching

<table>
<thead>
<tr>
<th>Table 4</th>
<th>Statistics of the improved prediction accuracy measured in PSNR (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
<td>Model $\to$</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>Table Tennis</td>
<td>Min</td>
</tr>
<tr>
<td></td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td>Average</td>
</tr>
<tr>
<td>Mobile Calendar</td>
<td>Min</td>
</tr>
<tr>
<td></td>
<td>Max</td>
</tr>
<tr>
<td></td>
<td>Average</td>
</tr>
</tbody>
</table>

$R$, $S$, $P$ and $D$ represent considering only the corresponding Lie operator individually. And $I_n$ represents applying integrated motion models iteratively with $n$ iterations.
window of (−15, +15) pixels is chosen for motion estimation. The improvement in accuracy can be measured in terms of the increase in the PSNR. As can be seen in Table 4, if only an individual Lie operator is allowed, on average, the largest improvement in prediction accuracy can be achieved by using the scaling operator for the “Table Tennis” sequence and the rotation operator for the “Mobile Calendar” sequence. Additionally, by applying the integrated motion models, a considerable amount of improvement in accuracy in motion estimation can be achieved. Using multiple iterations is also effective in further increasing the accuracy of motion estimation, albeit at the expense of extra computation efforts. Tests of the two sequences on a PC running Windows XP (with 3.40 GHz Pentium 4 CPU and 2GB RAM) show that, on average, estimating with individual motion model takes less than 1% of the time required for translation motion search using the full search method, whereas the integrated motion model search require about 1/5 of the time for the full search method.

4. Transformation estimation in a video codec

So far, we have discussed the prior work in an isolated test environment where motion is estimated and compensated using original reference frames, and have demonstrated that transformation estimation algorithms based on Lie derivatives can significantly improve the motion estimation accuracy achievable by using the conventional translation-only estimation. In this section, we evaluate the improvement of the prediction performance after our transformation estimation algorithms are integrated into an MPEG-2 codec, where prediction errors tend to propagate to succeeding frames. The results show that transformation estimation can lead to higher PSNR values for predicted MPEG-2 frames in two test sequences.

4.1. Embedding transformation estimation in an MPEG-2 codec

Fig. 6 illustrates an MPEG-2 encoder embedded with transformation estimation (TE). The module TE represents the iterative transformation estimation as shown in Fig. 5 (with four iterations for each block). The TE is applied on the prediction frame generated by the motion compensator. Here, P and the current frame (F2) are the inputs to the TE block. The outputs are a better-predicted frame PT and extra bits (as side information to code the θ values associated with the sequence of Lie operators found by the TE module). The new prediction error (between F2 and PT) will be used to replace the old prediction error between F2 and P in the conventional MPEG-2 codec. In the decoder (a version of which is also embedded in the encoder), we estimate each frame based on motion vectors (from the encoded bit stream), to obtain the prediction frame P and then apply TE on P to obtain the transformed frame PT. Here, PT and the transformation parameters (as a result of decoding the side information) are the two inputs for TE and the output is PT. Then PT and the prediction error will be added to reconstruct the original frame.

Fig. 6. An MPEG-2 encoder with transformation estimation.
In a practical video codec, motion estimation will be applied on a reconstructed previous frame rather than the original frame \( F_1 \). In MPEG-2, three types of pictures \( P \), \( B \), and \( I \) exist. \( P \)-pictures use only one reference picture, which is temporally located before the target picture, \( B \)-pictures use two references: one before and one after the target picture. And the third type of picture in MPEG-2, the so-called \( I \)-picture, is not motion-compensated. The composite motion model for TE can be applied irrespective of the order of the frames, because the algorithm is applied on the final predicted frame \( (P_T) \) as shown in Fig. 6.

### 4.2. Simulation results

In our simulations, \( \theta \) was chosen from the range \([-0.14, 0.14]\) with a step size of 0.02. We embedded the transformation estimation algorithms in the MPEG-2 codec. In the simulations, each GOP contains 12 frames with a structure of \( IBBBPPBBPBBBI \ldots \). Note that the rate of the compressed bit stream does not include the side information (extra bits for coding the \( \theta \) values).\(^3\)

Fig. 7 shows the predicted frames of second frame of “Table Tennis” sequence. After applying the transformation estimation algorithm, the improvement of the PSNR for the second predicted frame is 2.355 dB. Table 5 shows the maximum and average PSNR improvements achieved on these two test sequences under different bit rates. It is obvious from Table 5 that the transformation estimation method is valid for a wide range of bit rates. For the “Mobile Calendar” sequence, we observe an average PSNR improvement of about 1.4 dB, which is higher than about 1 dB improvement for the “Table Tennis” sequence. It can also be seen that with increased bit rates the improvement in PSNR also increases. Fig. 8 illustrates the PSNR improvement for 300 frames of the two sequences. Here, we can see that in one frame after every 12 frames there is no improvement in accuracy of estimation. This is because no transformation estimation was applied on those \( I \) frames.

### 4.3. Discussions

While motion models based on Lie operators can be used to effectively detect and compensate for small degrees of rotation, scaling and deformations of objects, we should emphasize that these models generally are not suitable for global motion estimation, where the assumption of very small degrees of transformations becomes invalid in most cases. As pointed out in Section 1.3, global motion estimation using motion models

\(^3\) See the discussions in Section 3.5.
has been proven to be effective in improving the accuracy of the motion estimation and compensation, however, more accurate local motion estimation is still needed to capture the local motions and transformations that could not be detected by the global motion estimation (due to outlier rejection), which gave rise to the well known two-stage approaches, with the local motion estimation applied following the global motion estimation [3,12,37]. Since the main focus of this paper is on local motion estimation, we did not use global motion estimation in the simulations. However, local transformation estimation applied on the block-by-block basis can still detect certain small degrees of global motions, which will have a bearing on the transformation parameters obtained for every block. Therefore, video sequences that exhibit relatively lower degree of global motion can give us a better idea of the effectiveness of the Lie operators method used solely as a local transform estimation approach. For example, the “Table Tennis” sequence contains much fewer global motions than the “Mobile Calendar” sequence, caused by, e.g., camera zooming. The influence of global motions can be seen in Fig. 8, which shows that, in general, there will be more PSNR improvements for the “Mobile Calendar” sequence than for the “Table Tennis” sequence. The same observation can also be made in Tables 4 and 5. Therefore, if the Lie operator based method is employed locally in conjunction with the global motion estimation, we can expect that local motion estimation is able to contribute to the overall increase of estimation accuracy by over 1 dB on average.

5. Conclusion

Conventional motion estimation algorithms are not suitable for capturing object transformations such as scaling, rotations and deformations in a video scene. To improve the accuracy of the conventional block-based local motion estimator using the translation-only motion model, we introduce two block-based methods using Lie operators, one for transformation estimation with an individual motion model and the other with composite model models. Simulation results demonstrate that remarkable improvements (of up to about 3.9 dB and about 1.4 dB on average) in estimation accuracy can be achieved by applying the transformation estimation approach with composite motion models. Furthermore, transformation estimation using Lie operators can lead to additional gains (about 1.4 dB) for predicted MPEG-2 frames of the two video sequences.

References


4 The panning of the cameras caused global translation motions in both sequences. However, these translation motions will be accounted for by the conventional translation motion estimation that precedes the transformation estimation based on Lie operators.


