frame. The prediction error, called the *displaced-frame difference* (DFD), is encoded instead of the current frame itself. The estimated motion information also has to be transmitted, unless the decoder can estimate it from previously decoded information. This section introduces the basics of motion estimation. It defines and formulates the motion estimation problem and describes the main approaches and models used to solve this problem. Examples of such solutions will be discussed in subsequent sections.

4.2.1 Projected Motion and Apparent Motion

In video, the 3-D motion of objects in space is projected as 2-D motion onto the image plane. This 2-D motion, called *projected motion*, is illustrated in Figure 4.1. Thus, motion estimation may refer to the process of estimating image-plane 2-D motion or object-space 3-D motion. Note that the two are not equivalent. In fact, 2-D motion estimation is usually the first step toward 3-D motion estimation. This chapter considers *2-D motion estimation* only. For 3-D motion estimation, the reader is referred to Ref. 10.

In video coding, motion is estimated by observing the spatiotemporal variation of intensity between frames. This is called the *apparent motion*. In the ideal case, apparent motion is equivalent to true projected motion. In practice, however, this is not always the case. For example, when a circle with uniform

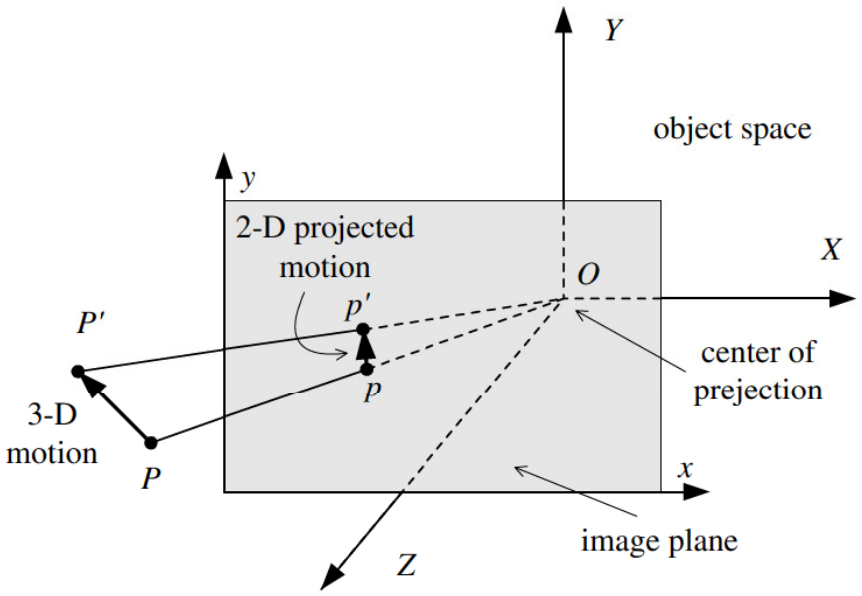


Figure 4.1: Projected motion

intensity rotates about its center, it has a rotational projected motion but zero apparent motion. Another example is a still object with change of illumination between frames. Although the object has zero projected motion, the change in illumination will result in some apparent motion. Hereafter, unless otherwise stated, the term *motion* will be used to refer to apparent motion rather than true projected motion.

Two-dimensional motion can be represented in terms of either 2-D displacement vectors, $\mathbf{d} = [d_x, d_y]^T$, or 2-D instantaneous velocity vectors, $\mathbf{v} = [v_x, v_y]^T = [\frac{dx}{dt}, \frac{dy}{dt}]^T$. A set of such vectors representing motion in a frame is called the *motion field* of the frame. The two representations are called the *displacement field* and the *velocity field* in the case of projected motion, or the *correspondence field* and the *optical flow field* in the case of apparent motion. However, in the video coding literature, it has become a convention to ignore this distinction and to use the terms *displacement field* and *velocity field* to refer to the apparent correspondence field and optical flow field, respectively. Hereafter, this convention will be adopted. Furthermore, this book uses the displacement field representation rather than the velocity field representation. Thus, the term *motion field* will always refer to the apparent correspondence field and the term *motion vector* will always refer to a displacement vector within this field.

4.2.2 Problem Formulation

Two-dimensional apparent motion can be attributed to three main causes. The first cause is global, or camera, motion. Even when there is no object motion in the frame, the motion of the camera induces a global motion. The second cause is local motion. This is the intrinsic motion of the objects in the scene. The third cause is illumination changes. Even when there is no object motion in the scene, changes in lighting conditions influence apparent motion.

All techniques considered in this chapter make no distinction between global and local motions, and they do not take into account illumination changes. Thus, they assume that global motion is taken into account through local motion and that the impact of illumination changes can be ignored. It should be pointed out, however, that some other techniques use a two-stage global/local motion estimation, e.g., Ref. 77, or estimate illumination changes, e.g., Ref. 78.

The 2-D apparent motion estimation problem can be formulated as a forward or a backward estimation problem depending on the temporal location of the reference frame with respect to the current frame.

In *backward motion estimation*, a pel $\mathbf{s} = [x, y]^T$ in the current frame at time t is related to a pel in a previous reference frame at time $t - \Delta t$ by

$$f_t(\mathbf{s}) = f_{t-\Delta t}(\mathbf{s} - \mathbf{d}(\mathbf{s})). \tag{4.1}$$

In *forward motion estimation*, however, the same pel is related to a pel in a future reference frame at time $t + \Delta t$ by

$$f_t(\mathbf{s}) = f_{t+\Delta t}(\mathbf{s} + \mathbf{d}(\mathbf{s})). \tag{4.2}$$

The aim of motion estimation is to find the motion vector $\mathbf{d}(\mathbf{s}) = [d_x(\mathbf{s}), d_y(\mathbf{s})]^T$. Note that $\mathbf{d}(\mathbf{s})$ is not necessarily a full-pel accurate motion vector. Thus, a motion estimation technique may need to access intensity values at nonsampling locations in the reference frame. This is achieved using interpolation techniques like nearest-neighbor, bilinear, and cubic interpolation. In this book, bilinear interpolation is employed because of its good compromise between interpolation quality and computational complexity. It is defined as

$$\begin{aligned} f(x, y) = {} & (1 - x_f)(1 - y_f) f(x_i, y_i) + x_f(1 - y_f) f(x_i + 1, y_i) \\ & + (1 - x_f) y_f f(x_i, y_i + 1) + x_f y_f f(x_i + 1, y_i + 1), \end{aligned} \tag{4.3}$$

where (x_i, y_i) and (x_f, y_f) are, respectively, the integer and fractional parts of the pel coordinates (x, y).

Care should be taken when interpreting the terms *forward* and *backward*. The two terms can be used to refer to either the motion estimation process or the motion compensation process. A forward motion estimation process corresponds to a backward motion compensation process, and vice versa. Note that forward motion estimation is associated with a coding delay. Thus, most video coding standards employ backward estimation (i.e., forward compensation), although forward estimation is sometimes employed (e.g., in B-frames in MPEG1–2 and PB-frames in H.263).

4.2.3 An Ill-Posed Problem

The preceding formulation of the motion estimation problem indicates that it is an ill-posed problem.[1] It suffers from the following problems [10]:

- Existence of solution: For example, no motion can be estimated for covered/uncovered background pels. This is known as the *occlusion problem*.
- Uniqueness of solution: At each pel, $\mathbf{s}$, the number of unknown independent variables (d_x and d_y) is twice the number of equations, (4.1) or (4.2). This is known as the *aperture problem*.

[1] A problem is called ill-posed if a unique solution does not exist and/or the solution does not continuously depend on the data [79].

- Continuity of solution: The motion estimate is highly sensitive to the presence of noise.

Because of this ill-posed nature of the problem, motion estimation algorithms use additional assumptions about the structure of the motion field. Such assumptions are referred to as *motion models*. They can be deterministic or probabilistic, parametric or nonparametric, as will be discussed in the following subsections.

4.2.4 Deterministic and Probabilistic Models

In a deterministic model, motion is seen as an unknown deterministic quantity. By maximizing the probability of the observed video sequence with respect to the unknown motion, this deterministic quantity can be estimated. The corresponding estimator is usually referred to as a *maximum likelihood* (ML) estimator. All motion estimation methods discussed in this chapter follow this deterministic approach.

In a probabilistic (or Bayesian) model, motion is seen as a random variable. Thus, the ensemble of motion vectors forms a random field. This field is usually modeled using a *Markov random field* (MRF). Given this model, motion estimation can be formulated as a *maximum a posteriori probability* (MAP) estimation problem. This problem can be solved using optimization techniques like simulated annealing, iterated conditional modes, mean field annealing, and highest confidence first. For a detailed description of Bayesian motion estimation methods, the reader is referred to Ref. 10.

4.2.5 Parametric and Nonparametric Models

In a parametric model, motion is represented by a set of *motion parameters*. Thus, the problem of motion estimation becomes a problem of estimating the motion parameters rather than the motion field itself. Since 2-D motion results from the projection of 3-D motion onto the image plane, a parametric 2-D motion model is usually derived from models describing 3-D motion, 3-D surfaces, and the projection geometry. For example, the assumptions of a planar 3-D surface moving in space according to a 3-D affine model and projected onto the image plane using an orthographic projection[2] results in a 2-D 6-parameter affine model. Different assumptions lead to different 2-D models. The 2-D models can be as complex as a quadratic 12-parameter model

[2]In an orthographic projection, it is assumed that all rays from a projected 3-D object to the image plane travel parallel to each other [10].

or as simple as a translational 2-parameter model (which is used in block-matching) [80]. Note that with parametric models, the constraint to regularize the ill-posed motion estimation problem is implicitly included in the motion model.

In nonparametric models, however, an explicit constraint (e.g., the smoothness of the motion field) is introduced to regularize the ill-posed problem of motion estimation.

4.2.6 Region of Support

An important parameter in motion estimation is the region of support. This is the set of pels to which the motion model applies. A region of support can be as large as a frame or as small as a single pel, it can be of fixed size or of variable size, and it can have a regular shape or an arbitrary shape.

Large regions of support result in a small motion overhead but may suffer from the *accuracy problem*. This means that pels within the region belong to different objects moving in different directions. Thus, the estimated motion parameters will not be accurate for some or all of the pels within the region.

The accuracy problem can be overcome by using small regions of support. This is, however, at the expense of an increase in motion overhead. Small support regions may also suffer from the *ambiguity problem*. This means that several patterns similar to the region may appear at multiple locations within the reference frame. This may lead to incorrect motion parameters.

4.3 Differential Methods

Differential methods are among the early approaches for estimating the motion of objects in video sequences. They are based on the relationship between the spatial and the temporal changes of intensity.

Differential methods were first proposed by Limb and Murphy in 1975 [81]. In their method, they use the magnitude of the temporal frame difference, FD, over a moving area, $\mathcal{A}$, to measure the speed of this area. To remove dependence on the area size, this measure is normalized by the horizontal, HD, or vertical, VD, spatial pel differences. Thus the estimated motion vector is given by

$$\hat{\mathbf{d}} = \begin{bmatrix} \hat{d}_x \\ \hat{d}_y \end{bmatrix} = \begin{bmatrix} \dfrac{\sum_{\mathbf{s}\in\mathcal{A}} \mathrm{FD}(\mathbf{s})\,\mathrm{sign}(\mathrm{HD}(\mathbf{s}))}{\sum_{\mathbf{s}\in\mathcal{A}} |\mathrm{HD}(\mathbf{s})|} \\ \dfrac{\sum_{\mathbf{s}\in\mathcal{A}} \mathrm{FD}(\mathbf{s})\,\mathrm{sign}(\mathrm{VD}(\mathbf{s}))}{\sum_{\mathbf{s}\in\mathcal{A}} |\mathrm{VD}(\mathbf{s})|} \end{bmatrix}, \tag{4.4}$$

where

$$\text{sign}(z) = \begin{cases} \frac{z}{|z|}, & \text{if } |z| \geq \text{threshold}, \\ 0, & \text{otherwise}, \end{cases} \tag{4.5}$$

$$\text{FD}(\mathbf{s}) = f_t(\mathbf{s}) - f_{t-\Delta t}(\mathbf{s}), \tag{4.6}$$

$$\text{HD}(\mathbf{s}) = \frac{1}{2}[f_t(x+1, y) - f_t(x-1, y)], \tag{4.7}$$

and

$$\text{VD}(\mathbf{s}) = \frac{1}{2}[f_t(x, y+1) - f_t(x, y-1)]. \tag{4.8}$$

The theoretical basis of differential methods were established later by Cafforio and Rocca in 1976 [82]. They start with the basic definition of the frame difference, Equation (4.6), and they rewrite it as

$$\begin{aligned} \text{FD}(\mathbf{s}) &= f_t(\mathbf{s}) - f_{t-\Delta t}(\mathbf{s}) \\ &= f_t(\mathbf{s}) - f_t(\mathbf{s} + \mathbf{d}). \end{aligned} \tag{4.9}$$

For small values of $\mathbf{d}$, the right-hand side of Equation (4.9) can be replaced by its Taylor series expansion about $\mathbf{s}$, as follows:

$$\text{FD}(\mathbf{s}) = -\mathbf{d}^T \nabla_{\mathbf{s}} f_t(\mathbf{s}) + \text{higher-order terms}, \tag{4.10}$$

where $\nabla_{\mathbf{s}} = [\frac{\partial}{\partial x}, \frac{\partial}{\partial y}]^T$ is the spatial gradient with respect to $\mathbf{s}$. Ignoring the higher-order terms and assuming that motion is constant over an area $\mathcal{A}$, linear regression can be used to obtain the minimum mean square estimate of $\mathbf{d}$ as

$$\hat{\mathbf{d}} = -\left[\sum_{\mathbf{s}\in\mathcal{A}} \nabla_{\mathbf{s}} f_t(\mathbf{s}) \nabla_{\mathbf{s}}^T f_t(\mathbf{s})\right]^{-1} \left[\sum_{\mathbf{s}\in\mathcal{A}} \text{FD}(\mathbf{s}) \nabla_{\mathbf{s}} f_t(\mathbf{s})\right]. \tag{4.11}$$

Note that this equation is highly dependent on the spatial gradient, $\nabla_{\mathbf{s}}$. For this reason, differential methods are also known as *gradient methods*. Using the approximation $\nabla_{\mathbf{s}} f_t(\mathbf{s}) \approx [\text{HD}(\mathbf{s}), \text{VD}(\mathbf{s})]^T$, Equation (4.11) reduces to

$$\begin{aligned} \hat{\mathbf{d}} = -&\begin{bmatrix} \sum_{\mathbf{s}\in\mathcal{A}} \text{HD}^2(\mathbf{s}) & \sum_{\mathbf{s}\in\mathcal{A}} \text{HD}(\mathbf{s}) \cdot \text{VD}(\mathbf{s}) \\ \sum_{\mathbf{s}\in\mathcal{A}} \text{HD}(\mathbf{s}) \cdot \text{VD}(\mathbf{s}) & \sum_{\mathbf{s}\in\mathcal{A}} \text{VD}^2(\mathbf{s}) \end{bmatrix}^{-1} \\ &\times \begin{bmatrix} \sum_{\mathbf{s}\in\mathcal{A}} \text{FD}(\mathbf{s}) \cdot \text{HD}(\mathbf{s}) \\ \sum_{\mathbf{s}\in\mathcal{A}} \text{FD}(\mathbf{s}) \cdot \text{VD}(\mathbf{s}) \end{bmatrix}. \end{aligned} \tag{4.12}$$

By ignoring the cross terms (i.e., $\sum_{\mathbf{s}\in\mathcal{A}} \mathrm{HD}(\mathbf{s})\cdot\mathrm{VD}(\mathbf{s}) \approx 0$), it can be shown that the general analytical solution of Cafforio and Rocca (Equation (4.12)) reduces to the simple heuristic solution of Limb and Murphy (Equation (4.4)).

The main assumption in deriving the differential estimate of Equation (4.12) using Taylor series expansion is that the motion vector $\mathbf{d}$ is small. As $\mathbf{d}$ increases, the quality of the approximation becomes poor. Thus, the main drawback of differential methods is that they can only be used to measure small motion displacements (up to about ± 3 pels). A number of methods have been proposed to overcome this problem, like, for example, the iterative method of Yamaguchi [83]. In this method, an initial motion vector is first estimated, using Equation (4.12), between a block in the current frame and a corresponding block in the same location in the reference frame. In the next iteration, the position of the matched block in the reference frame is shifted by the initial motion vector, and then the differential method is applied again to produce a second estimate. This second estimate acts as a correction term for the initial estimate. This process of shift and estimation continues until the correction term becomes adequately small.

Another drawback of differential methods is that the spatial gradient operator, $\nabla_{\mathbf{s}}$, is sensitive to data noise. This can be reduced by using a larger set of data in its calculation.

There are also cases where differential methods can fail [84]. For example, in smooth areas the gradient is approximately equal to zero and the matrix in Equation (4.12) becomes singular. Also, when motion is parallel to edges in the image, i.e., $\mathbf{d}^T\nabla_{\mathbf{s}} \approx 0$, the frame difference, Equation (4.10), becomes zero, giving a wrong displacement of zero. Such problems may be partially solved by increasing the data area, but this may give rise to the accuracy problem.

4.4 Pel-Recursive Methods

Given a function $g(\mathbf{r})$ of several unknowns $\mathbf{r} = [r_1, \ldots, r_n]^T$, the most straightforward way to minimize it is to calculate its partial derivatives with respect to each unknown, set them equal to 0, and solve the resulting simultaneous equations. This is called *gradient-based optimization* and can be represented in vector form as

$$\nabla_{\mathbf{r}} g(\mathbf{r}) = 0. \tag{4.13}$$

In cases where the function $g(\mathbf{r})$ cannot be represented in closed form and/or the set of simultaneous Equations (4.13) cannot be solved, numerical iterative methods are employed.

One of the simplest numerical methods is the steepest-descent method. Since the gradient vector points in the direction of the maximum, this method updates

the present estimate, $\hat{\mathbf{r}}^i$, of the location of the minimum in the direction of the negative gradient, to obtain a new improved estimate

$$\hat{\mathbf{r}}^{i+1} = \hat{\mathbf{r}}^i - \alpha \nabla_{\mathbf{r}} g(\hat{\mathbf{r}}^i), \tag{4.14}$$

where $\alpha > 0$ is an update step size and i is the iteration index.

Pel-recursive methods are based on an iterative gradient-based minimization of the prediction error. They were first proposed by Netravali and Robbins in 1979 [85]. In their algorithm, they use a steepest-descent approach to iteratively minimize the square of the displaced-frame difference, $\mathrm{DFD}(\mathbf{s}, \mathbf{d})$, with respect to the displacement vector, $\mathbf{d}$. Thus

$$g(\mathbf{r}) = \mathrm{DFD}^2(\mathbf{s}, \mathbf{d}), \tag{4.15}$$

where

$$\mathrm{DFD}(\mathbf{s}, \mathbf{d}) = f_t(\mathbf{s}) - f_{t-\Delta t}(\mathbf{s} - \mathbf{d}). \tag{4.16}$$

Substituting Equation (4.15) into Equation (4.14) and setting $\alpha = \frac{\varepsilon}{2}$ gives

$$\hat{\mathbf{d}}^{i+1} = \hat{\mathbf{d}}^i - \tfrac{\varepsilon}{2} \nabla_{\mathbf{d}} \mathrm{DFD}^2(\mathbf{s}, \hat{\mathbf{d}}^i). \tag{4.17}$$

Now,

$$\begin{aligned} \nabla_{\mathbf{d}} \mathrm{DFD}^2(\mathbf{s}, \mathbf{d}) &= 2\,\mathrm{DFD}(\mathbf{s}, \mathbf{d})\, \nabla_{\mathbf{d}} \mathrm{DFD}(\mathbf{s}, \mathbf{d}) \\ &= 2\,\mathrm{DFD}(\mathbf{s}, \mathbf{d})\, \nabla_{\mathbf{d}} [f_t(\mathbf{s}) - f_{t-\Delta t}(\mathbf{s} - \mathbf{d})] \\ &= 2\,\mathrm{DFD}(\mathbf{s}, \mathbf{d})\, \nabla_{\mathbf{s}} f_{t-\Delta t}(\mathbf{s} - \mathbf{d}). \end{aligned} \tag{4.18}$$

Substituting Equation (4.18) into Equation (4.17) gives

$$\hat{\mathbf{d}}^{i+1} = \hat{\mathbf{d}}^i - \varepsilon \mathrm{DFD}(\mathbf{s}, \hat{\mathbf{d}}^i) \nabla_{\mathbf{s}} f_{t-\Delta t}(\mathbf{s} - \hat{\mathbf{d}}^i), \tag{4.19}$$

where the spatial gradient $\nabla_{\mathbf{s}} f_{t-\Delta t}(\mathbf{s} - \hat{\mathbf{d}}^i)$ can be approximated by Equations (4.7) and (4.8) but evaluated at a displaced location $(\mathbf{s} - \mathrm{NINT}[\hat{\mathbf{d}}^i])$ in the reference frame. As in differential methods, this estimate is highly dependent on the spatial gradient. For this reason, pel-recursive methods are sometimes considered a subset of gradient or differential methods.

The iterative approach of Equation (4.19) is normally applied on a pel-by-pel basis, leading to a *dense* motion field, $\hat{\mathbf{d}}(\mathbf{s})$. Iterations may proceed along a scanning line, from line to line, or from frame to frame. In order to smooth out the effect of noise, the update term can be evaluated over an area $\mathcal{A} = \{\mathbf{s}_1, \ldots, \mathbf{s}_p\}$ as follows:

$$\hat{\mathbf{d}}^{i+1} = \hat{\mathbf{d}}^i - \varepsilon \sum_{j=1}^{p} W_j \,\mathrm{DFD}(\mathbf{s}_j, \hat{\mathbf{d}}^i) \nabla_{\mathbf{s}} f_{t-\Delta t}(\mathbf{s}_j - \hat{\mathbf{d}}^i), \tag{4.20}$$

where $W_j \geq 0$ and $\sum_{j=1}^{p} W_j = 1$. Netravali and Robbins also proposed a simplified expression for hardware implementation:

$$\hat{\mathbf{d}}^{i+1} = \hat{\mathbf{d}}^{i} - \varepsilon \operatorname{sign}[\mathrm{DFD}(\mathbf{s}, \hat{\mathbf{d}}^{i})] \operatorname{sign}[\nabla_{\mathbf{s}} f_{t-\Delta t}(\mathbf{s} - \hat{\mathbf{d}}^{i})]. \tag{4.21}$$

The convergence of this method is highly dependent on the constant step size ε. A high value of ε leads to quick convergence but less accuracy, whereas a small value of ε leads to slower convergence but more accurate estimates. Thus, a compromise between the two is desired. A number of algorithms have been reported to improve the performance of pel-recursive algorithms, e.g., Ref. 86. Most of them are based on the idea of substituting the constant step size ε by a variable step size to achieve better adaptation to the local image statistics and, consequently, faster convergence and higher accuracy. A good review of such methods with comparative results can be found in Ref. 87.

The dense motion field of pel-recursive methods can overcome the accuracy problem. This is, however, at the expense of a large motion overhead. To overcome this drawback, the update term from one iteration to the other can be based on previously transmitted data only. In this case, the decoder can estimate the same displacements generated at the encoder, and no motion information needs to be transmitted. A disadvantage of this causal approach, however, is that it constrains the method and reduces its prediction capability. In addition, it increases the complexity of the decoder.

Another disadvantage of pel-recursive methods is that they can easily converge to local minima within the error surface. In addition, smooth intensity regions, discontinuities within the motion field, and large displacements cannot be efficiently handled [55].

4.5 Frequency-Domain Methods

Frequency-domain motion estimation methods are based on the *Fourier transform* (FT) property that a translational displacement in the spatial domain corresponds to a linear phase shift in the frequency domain. Thus, assuming that the image intensities of the current frame, f_t, and the reference frame, $f_{t-\Delta t}$, differ over a moving area, $\mathcal{A}$, only due to a translational displacement, (d_x, d_y), then

$$f_t(x, y) = f_{t-\Delta t}(x - d_x, y - d_y), \quad (x, y) \in \mathcal{A}. \tag{4.22}$$

Taking the FT of both sides with respect to the spatial variables (x, y) gives the following frequency-domain equation in the frequency variables (w_x, w_y):

$$F_t(w_x, w_y) = F_{t-\Delta t}(w_x, w_y) e^{j(-w_x d_x - w_y d_y)}, \tag{4.23}$$

where F_t and $F_{t-\Delta t}$ are the FTs of the current and reference frames, respectively. In Ref. 88, Haskell noticed this relationship but did not propose an algorithm to recover the displacement from the phase shift.

If we define $\Delta\phi(w_x, w_y)$ as the phase difference between the FT of the current frame and that of the reference frame, then

$$\begin{aligned} e^{j\Delta\phi(w_x,w_y)} &= e^{j[\phi_t(w_x,w_y)-\phi_{t-\Delta t}(w_x,w_y)]} \\ &= e^{j\phi_t(w_x,w_y)} \cdot e^{-j\phi_{t-\Delta t}(w_x,w_y)} \\ &= \frac{F_t(w_x,w_y)}{|F_t(w_x,w_y)|} \cdot \frac{F^*_{t-\Delta t}(w_x,w_y)}{|F^*_{t-\Delta t}(w_x,w_y)|}, \end{aligned} \tag{4.24}$$

where ϕ_t and $\phi_{t-\Delta t}$ are the phase components of F_t and $F_{t-\Delta t}$, respectively, and the superscript $*$ indicates the complex conjugate. If we define $c_{t,t-\Delta t}(x, y)$ as the inverse FT of $e^{j\Delta\phi(w_x,w_y)}$, then

$$\begin{aligned} c_{t,t-\Delta t}(x,y) &= \mathcal{F}^{-1}\{e^{j\Delta\phi(w_x,w_y)}\} \\ &= \mathcal{F}^{-1}\{e^{j\phi_t(w_x,w_y)} \cdot e^{-j\phi_{t-\Delta t}(w_x,w_y)}\} \\ &= \mathcal{F}^{-1}\{e^{j\phi_t(w_x,w_y)}\} \otimes \mathcal{F}^{-1}\{e^{-j\phi_{t-\Delta t}(w_x,w_y)}\}, \end{aligned} \tag{4.25}$$

where $\otimes$ is the 2-D convolution operation. In other words, $c_{t,t-\Delta t}(x, y)$ is the cross-correlation of the inverse FTs of the phase components of F_t and $F_{t-\Delta t}$. For this reason, $c_{t,t-\Delta t}(x, y)$ is known as the *phase correlation function*. The importance of this function becomes apparent if it is rewritten in terms of the phase difference in Equation (4.23):

$$\begin{aligned} c_{t,t-\Delta t}(x,y) &= \mathcal{F}^{-1}\{e^{j\Delta\phi(w_x,w_y)}\} \\ &= \mathcal{F}^{-1}\{e^{j(-w_x d_x - w_y d_y)}\} \\ &= \delta(x-d_x, y-d_y). \end{aligned} \tag{4.26}$$

Thus, the phase correlation surface has a distinctive impulse at (d_x, d_y). This observation is the basic idea behind the *phase correlation* motion estimation method. In this method, Equation (4.24) is used to calculate $e^{j\Delta\phi(w_x,w_y)}$, the inverse FT is then applied to obtain $c_{t,t-\Delta t}(x, y)$, and the location of the impulse in this function is detected to estimate (d_x, d_y).

In practice, the impulse in the phase correlation function degenerates into one or more peaks. This is due to many factors, like the use, in digital images, of the discrete Fourier transform (DFT) instead of the FT, the presence of more than one moving object within the considered area $\mathcal{A}$, and the presence of

noise. In particular, the use of the 2-D DFT instead of the 2-D FT results in the following effects [10]:

- *The boundary effect*: In order to obtain a perfect impulse, the translational displacement must be cyclic. In other words, objects disappearing at one end of the moving area must reappear at the other end. In practice this does not happen, which leads to the degeneration of the impulse into peaks. Furthermore, the DFT assumes periodicity in both directions. In practice, however, discontinuities occur from left to right and from top to bottom, introducing spurious peaks.
- *Spectral leakage*: In order to obtain a perfect impulse, the translational displacement must correspond to an integer multiple of the fundamental frequency. In practice, noninteger motion vectors may not satisfy this condition, leading to the well-known spectral leakage phenomenon [89], which degenerates the impulse into peaks.
- *Displacement wrapping*: The 2-D DFT is periodic with the area size (N_x, N_y). Negative estimates will be wrapped and will appear as positive displacements. To accommodate negative displacements, the estimated displacement needs to be unwrapped as follows [10]:

$$\hat{d}_i = \begin{cases} \hat{d}_i & \text{if } |\hat{d}_i| \leq \frac{N_i}{2} \text{ and } N_i \text{ is even} \\ & \text{or if } |\hat{d}_i| \leq \frac{N_i - 1}{2} \text{ and } N_i \text{ is odd,} \\ \hat{d}_i - N_i, & \text{otherwise.} \end{cases} \tag{4.27}$$

  This means that the range of estimates is limited to $[\frac{-N_i}{2} + 1, \frac{N_i}{2}]$ for N_i even.

The phase correlation motion estimation method was first reported by Kuglin and Hines in 1975 [90]. It was later extensively studied by Thomas [91]. In his study, Thomas analyzed the properties of the phase correlation function. He suggested using a weighting function to smooth the correlation surface and suppress spurious peaks. He also proposed a second stage to the method, in which smaller moving areas are used and more than one dominant peak from the first stage are considered and compared. Girod [92] augmented this by a third stage, in which the estimated integer-pel motion displacement is refined to subpel accuracy.

The phase correlation method has a number of desirable properties. It has a small computational complexity, especially with the use of *fast Fourier transforms* (FFTs). In addition, it is relatively insensitive to illumination changes because shifts in the mean value or multiplication by a constant do not affect the Fourier phase. Furthermore, the method can detect multiple moving objects,

because they appear as multiple peaks in the correlation surface. In addition to its use in video coding, the phase correlation method has been successfully incorporated into commercial standards conversion equipment [93].

There are few other frequency-domain motion estimation methods. For example, Chou and Hang [94] analyzed frequency-domain motion estimation in both noise-free and noisy situations. Their analysis is very similar to the noise analysis in phase or frequency modulation systems, and it provides insights into the performance limits of motion estimation. They formulated frequency-domain motion estimation as a set of simultaneous equations, which they solved using a modified *least-mean-square* (LMS) algorithm. The resulting algorithm is known as the *frequency component method*. It provides more reliable estimates than the phase correlation method, particularly for noisy sequences. Young and Kingsbury [95] proposed a frequency-domain method based on the complex lapped transform. Koc and Liu [96] used the pseudophase hidden in the DCT transform to propose a DCT-based frequency-domain motion estimation method. The algorithm has a low computational complexity and was later extended to achieve interpolation-free subpel accuracy [97].

4.6 Block-Matching Methods

Block-matching motion estimation (BMME) is the most widely used motion estimation method for video coding. Interest in this method was initiated by Jain and Jain in 1981 [54]. In their *block-matching algorithm* (BMA), the current frame, f_t, is first divided into blocks of $M \times N$ pels. The algorithm then assumes that all pels within the block undergo the same translational movement. Thus, the same motion vector, $\mathbf{d} = [d_x, d_y]^T$, is assigned to all pels within the block. This motion vector is estimated by searching for the *best-match* block in a larger *search window* of $(M + 2d_{m_x}) \times (N + 2d_{m_y})$ pels centered at the same location in a reference frame, $f_{t-\Delta t}$, where d_{m_x} and d_{m_y} are the maximum allowed motion displacements in the horizontal and vertical directions, respectively. This process is illustrated in Figure 4.2 and can be formulated as follows:

$$(\hat{d}_x, \hat{d}_y) = \arg \min_{i,j} \text{BDM}(i,j), \text{ where } |i| \leq d_{m_x} \text{ and } |j| \leq d_{m_y}, \tag{4.28}$$

and $\text{BDM}(i,j)$ is a *block distortion measure* that measures the quality of match between the block in the current frame and a corresponding candidate block in the reference frame shifted by a displacement (i,j). It is very common to use square blocks of $N \times N$ pels and a maximum motion displacement of $\pm d_m$ in both directions. When Equation (4.28) is evaluated for all possible (i,j)

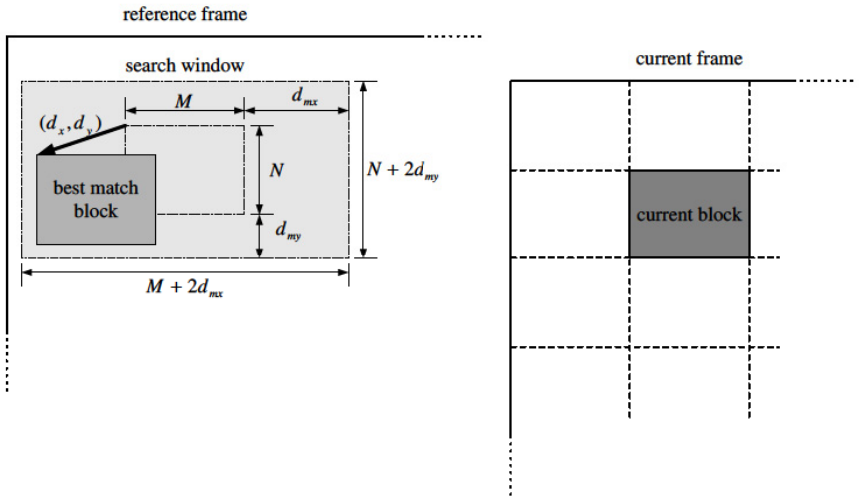


Figure 4.2: Block-matching motion estimation

displacements (i.e., for all possible candidate blocks in the search window), the BMA is referred to as the *full-search* (FS) algorithm.

Since its introduction, BMME has attracted considerable attention, and many refinements to the basic BMA have been proposed. In the following subsections, different parameters of the BMA are introduced and their impact on performance is evaluated. A number of refinements to the basic BMA are also examined.

4.6.1 Matching Function

The matching function (or the BDM) can be any function that measures the distortion or the match between the block, $\mathscr{B}$, in the current frame and the displaced candidate block in the reference frame. The choice of a suitable BDM is very important, for it impacts both the prediction quality and the computational complexity of the algorithm.

One possible matching function is the *normalized cross-correlation function*[3] (NCCF), defined as

$$\mathrm{NCCF}(i,j) = \frac{\sum_{(x,y)\in\mathscr{B}} f_t(x,y)\cdot f_{t-\Delta t}(x-i,y-j)}{\sqrt{\sum_{(x,y)\in\mathscr{B}} f_t^2(x,y)}\cdot\sqrt{\sum_{(x,y)\in\mathscr{B}} f_{t-\Delta t}^2(x-i,y-j)}}. \quad (4.29)$$

[3]The NCCF is a measure of the correlation between two blocks rather than the distortion between them. Thus, when used in BMA, the minimization process in Equation (4.28) becomes a maximization process.

Since the motion estimation process aims at minimizing the DFD signal, a natural choice for the matching function is the *mean squared error*, which is often formulated as the *sum of squared differences* (SSD):

$$\mathrm{SSD}(i,j) = \sum_{(x,y)\in\mathcal{B}} (f_t(x,y) - f_{t-\Delta t}(x-i, y-j))^2. \tag{4.30}$$

A very similar matching function is the *sum of absolute differences* (SAD):

$$\mathrm{SAD}(i,j) = \sum_{(x,y)\in\mathcal{B}} |f_t(x,y) - f_{t-\Delta t}(x-i, y-j)|. \tag{4.31}$$

To compare the performance of these matching functions, a full-pel full-search BMA was implemented. The algorithm uses 16×16 blocks and a maximum allowed motion displacement of ± 15 pels in both directions. In this algorithm, motion is estimated and compensated using original previous frames, and motion vectors are restricted so that they do not point outside the reference frame. Motion vectors are encoded using the median predictor and the VLC table of the H.263 standard. Unless otherwise stated, all subsequent results in this chapter use the same simulation conditions. Figure 4.3 compares the performances of the algorithm with different matching functions when applied to the first 10 frames of the FOREMAN sequence at a frame rate of 8.33 frames/s (i.e., a frame skip[4] of 3). The quoted PSNR values are for the luma component only. It can be seen from this figure that the SSD measure achieves the best performance, followed very closely by the SAD measure. The NCCF measure, on the other hand, has the worst performance. While Figure 4.3 compares the performance in terms of prediction quality, Table 4.1 compares the performances in terms of computational complexity. It can be seen that the SAD measure has the lowest computational complexity, because it involves no multiplications. Because of its good prediction quality and small computational complexity, SAD is preferred by most implementations. All subsequent results assume the use of SAD as the matching function.

There are many other proposed matching functions. Most of them attempt to further reduce complexity, but this is often at the expense of a reduced prediction quality. A more detailed discussion of such functions is deferred to Chapter 7.

[4]Throughout this book, the term *frame skip* will be used to quantify the amount of temporal subsampling with respect to the original frame rate. For example, a frame skip of 3 means that the original sequence is temporally subsampled by a factor of 3:1. Thus, if the original sequence has a frame rate of 30 frames/s, then the subsampled sequence will have a frame rate of $30/3 = 10$ frames/s.

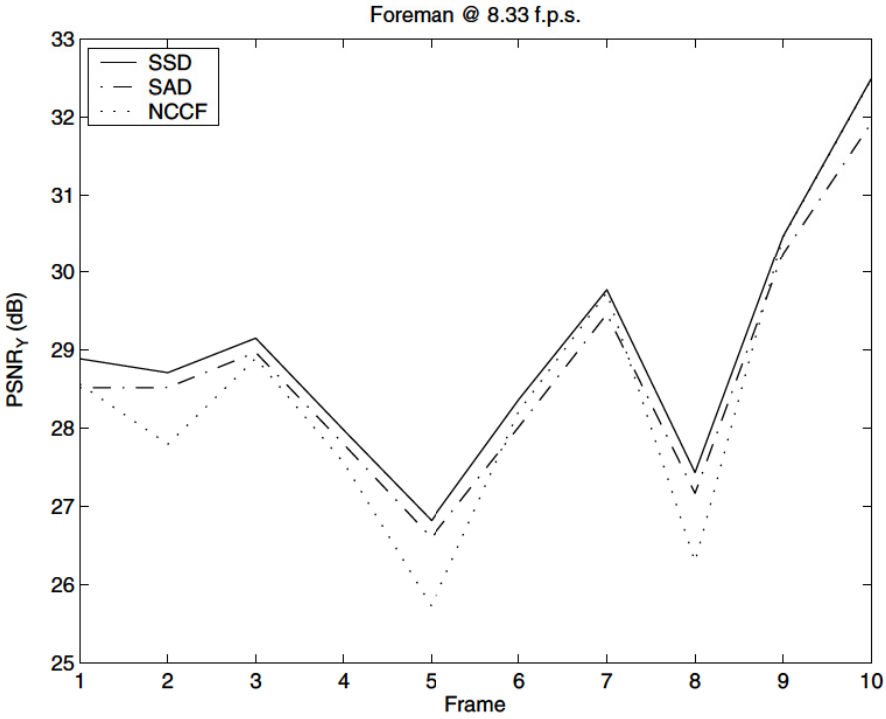


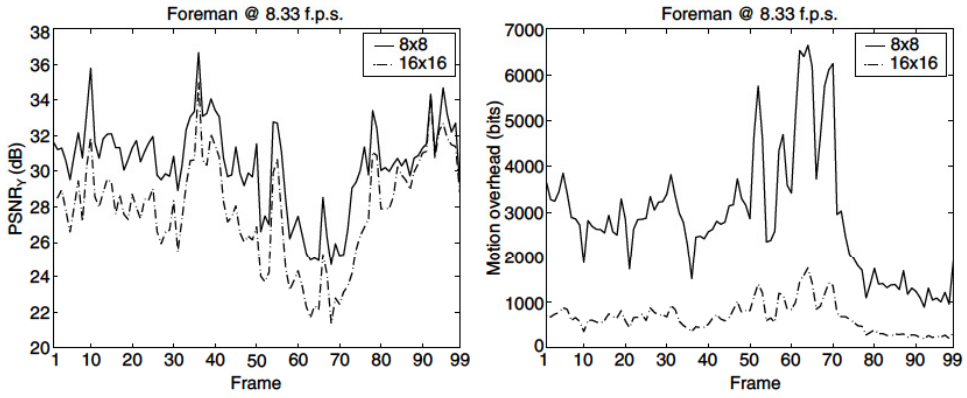
Figure 4.3: Reconstruction quality of SSD, SAD, and NCCF

Table 4.1: Computational complexity of SSD, SAD, and NCCF for an $N \times N$ block

	SAD	SSD	NCCF
$\lvert \cdot \rvert$	N^2	–	–
$-$	N^2	N^2	-
$+$	$N^2 - 1$	$N^2 - 1$	$3(N^2 - 1)$
$\times$	–	N^2	$3N^2 + 1$
$\div$	–	–	1
$\sqrt{}$	–	–	2

4.6.2 Block Size

Another important parameter of the BMA is the block size. Figure 4.4 shows the performance of the BMA with two different sizes, 8×8 and 16×16. It can be seen in Figure 4.4(a) that a smaller block size achieves better prediction quality. This is due to a number of reasons. A smaller block size reduces the effect of the accuracy problem. In other words, with a smaller block size, there is less possibility that the block will contain different objects moving in



(a) Prediction quality (b) Motion overhead

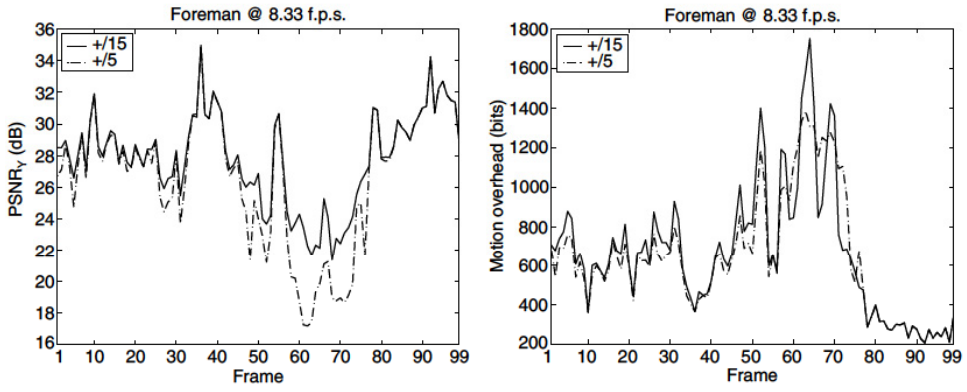
Figure 4.4: Performance of the BMA with different block sizes

different directions. In addition, a smaller block size provides a better piecewise translational approximation to nontranslational motion. Since a smaller block size means that there are more blocks (and consequently more motion vectors) per frame, this improved prediction quality comes at the expense of a larger motion overhead, as can be seen in Figure 4.4(b). Most video coding standards use a block size of 16×16 as a compromise between prediction quality and motion overhead. A number of variable-block-size motion estimation methods have also been proposed in the literature [98, 99]. As already discussed, the advanced prediction mode of the H.263 standard allows adaptive switching between block sizes of 16×16 and 8×8 on an MB basis.

4.6.3 Search Range

The maximum allowed motion displacement d_m, also known as the *search range*, has a direct impact on both the computational complexity and the prediction quality of the BMA. A small d_m results in poor compensation for fast-moving areas and consequently poor prediction quality. This is evident from Figure 4.5(a), which compares the performance of two ranges, ± 5 and ± 15. A large d_m, on the other hand, results in better prediction quality but leads to an increase in the computational complexity (since there are $(2d_m+1)^2$ possible blocks to be matched in the search window). A larger d_m can also result in longer motion vectors and consequently a slight increase in motion overhead,[5] as can be seen from Figure 4.5(b). In general, a maximum allowed

[5] As will be shown later, in block-motion fields, larger displacements are, in general, less probable. Thus, most video codecs assign longer codewords for longer motion vectors.



(a) Prediction quality (b) Motion overhead

Figure 4.5: Performance of the BMA with different search ranges

displacement of $d_m = \pm 15$ pels is sufficient for low-bit-rate applications. As already discussed, the H.263 standard uses a maximum displacement of about ± 15 pels, although this range can optionally be doubled with the unrestricted motion vector mode.

4.6.4 Search Accuracy

Initially, the BMA was designed to estimate motion displacements with full-pel accuracy. Clearly, this limits the performance of the algorithm, since in reality the motion of objects is completely unrelated to the sampling grid. A number of workers in the field have proposed to extend the BMA to subpel accuracy. For example, Ericsson [100] demonstrated that a prediction gain of about 2 dB can be obtained by moving from full-pel to 1/8-pel accuracy. Girod [92] presented an elegant theoretical analysis of motion-compensating prediction with subpel accuracy. He termed the resulting prediction gain the *accuracy effect*. He also showed that there is a "critical accuracy" beyond which the possibility of further improving prediction is very small. He concluded that with block sizes of 16×16, quarter-pel accuracy is desirable for broadcast TV signals, whereas half-pel accuracy appears to be sufficient for videophone signals. Today, most video coding standards adopt subpel accuracy in its half-pel form. In fact, it has been shown [65] that most of the performance gain of H.263 over H.261 can be attributed to the move from full-pel to half-pel accuracy.

It should be pointed out, however, that the improved prediction quality of subpel accuracy comes at the expense of a significant increase in computational complexity. This increase is due to two reasons. First, the reference frame intensities have to be interpolated at subpel locations. Second, there are now

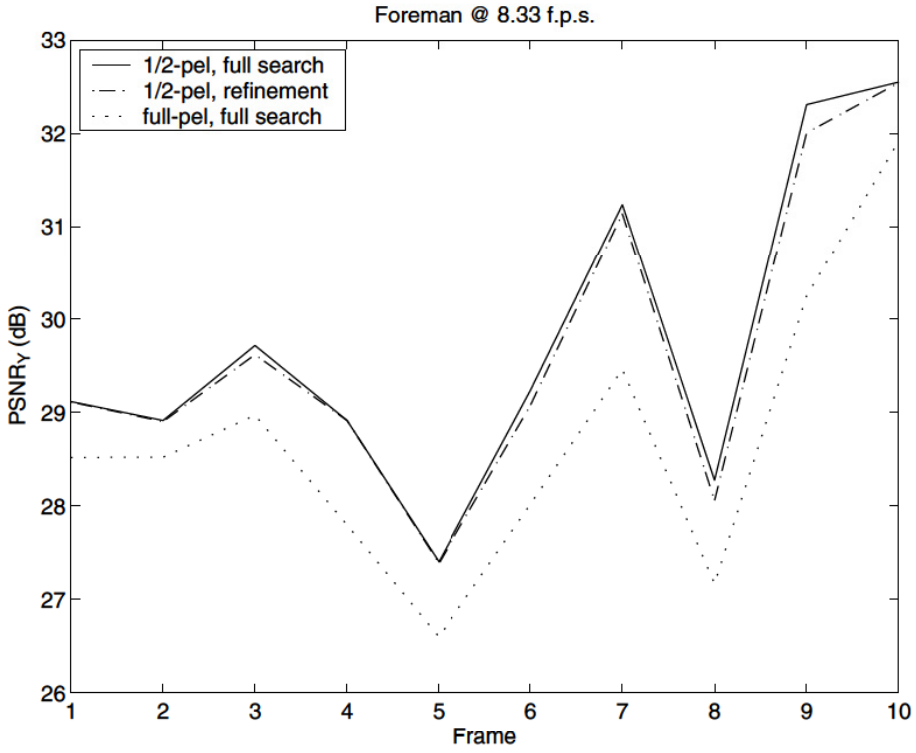


Figure 4.6: Performance of the BMA with subpel accuracy

more possible candidate blocks within the search window. For example, when moving from full-pel to half-pel accuracy, the number of candidate blocks in the search window increases from $(2d_m + 1)^2$ to $(4d_m + 1)^2$. To alleviate this complexity, most video codecs implement subpel accuracy as a postprocessing stage, where first a full-pel motion vector is obtained, usually using full search, and then this vector is *refined* to subpel accuracy using a limited search. This provides a large saving in computational complexity and at the same time maintains the improved prediction quality, as can be seen in Figure 4.6.

4.6.5 Unrestricted Motion Vectors

In some cases (like, for example, in border blocks) part of the search window is outside the reference frame area. This means that some of the candidate blocks in the search window are either partially or completely out of the reference frame. There are two ways to handle such candidate blocks. In the *restricted motion vectors* method, such blocks are ignored and skipped during motion estimation. In the *unrestricted motion vectors* method, however, such

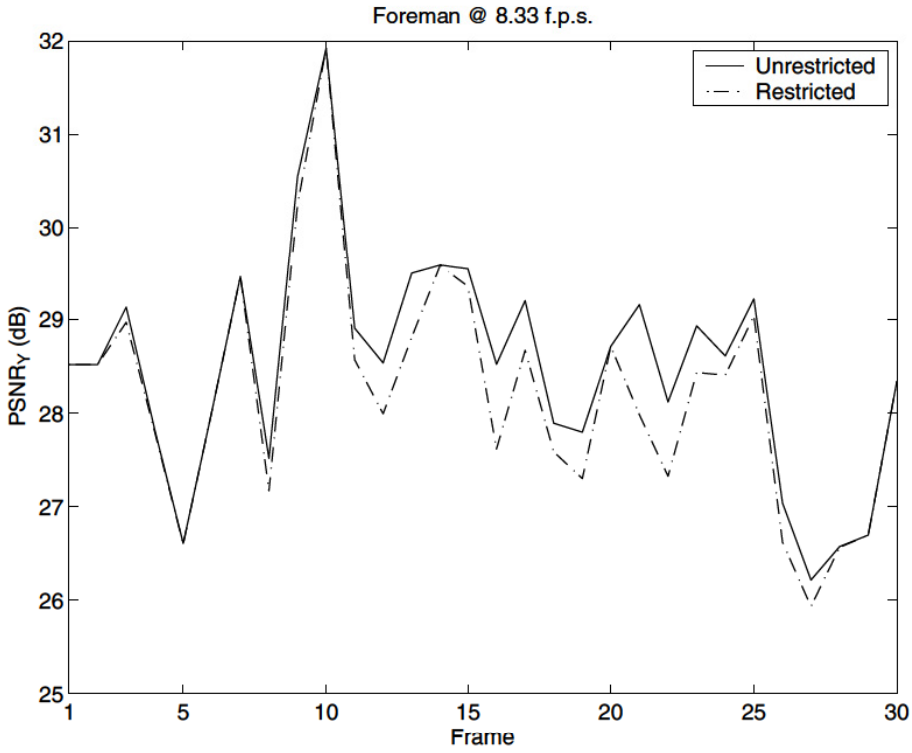


Figure 4.7: Performance of the BMA with restricted and unrestricted motion vectors

blocks are included in the motion estimation and compensation process. In this case, a referenced pel outside the frame is usually approximated by the closest border pel. This unrestricted method can improve the prediction quality along frame borders, especially in cases of camera or background movement. This is particularly useful in small frame formats, where border blocks represent a high percentage of the frame area. Figure 4.7 illustrates this improvement for part of the FOREMAN sequence. The method is included in the H.263 optional unrestricted motion vector mode and also in the advanced prediction mode.

4.6.6 Overlapped Motion Compensation

As already discussed, the BMA assumes that each block of pels moves with a uniform translational motion. Because this assumption does not always hold true, the method is known to produce blocking artefacts in the reconstructed frames. One method that reduces this effect is *overlapped motion compensation* (OMC). The method was first proposed by Watanabe and Singhal in 1991 [101]. In BMA, the estimated block motion vector is used to copy a displaced

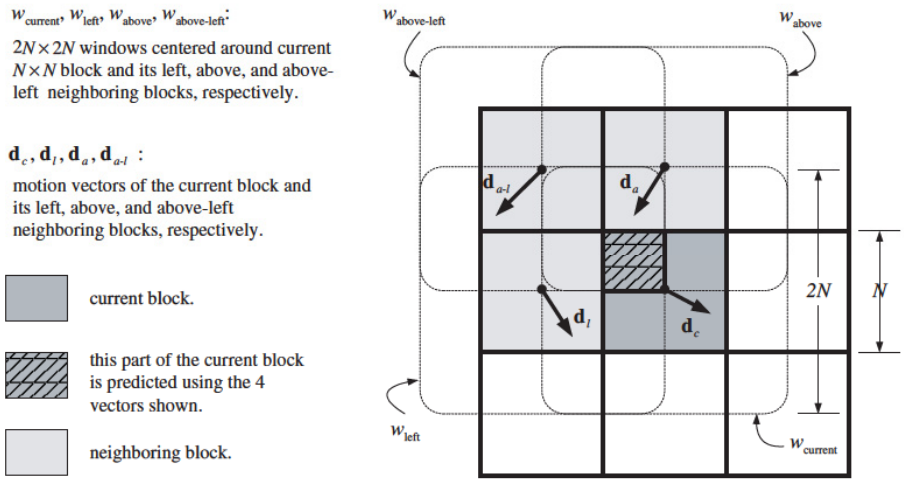


Figure 4.8: Overlapped motion compensation for the top-left quadrant of the current block

$N \times N$ block from the reference frame to the current $N \times N$ block in the current frame. In OMC, however, the estimated block motion vector is used to copy a larger block (say, $2N \times 2N$) from the reference frame to a position centered around the current $N \times N$ block. As illustrated in Figure 4.8, since they are larger than the compensated blocks, the copied blocks overlap, hence the name *overlapped motion compensation*. Each copied block is weighted by a smooth window, with higher weights at the center and lower weights toward the borders. This means that the estimated motion vector is given more influence in the center of the block, and this influence decays toward the borders, where neighboring motion vectors start taking over. This ensures a smooth transition between blocks and therefore reduces blocking artefacts. Overlapped motion estimation and compensation can also be implemented in the frequency domain, as proposed by Young and Kingsbury [95].

Another view of the OMC process is that each pel in the current $N \times N$ block is compensated using more than one motion vector. For example, in Figure 4.8, each pel is compensated using four motion vectors. The set of motion vectors is decided according to the spatial position of the pel within the block. A pel in the top-left quadrant of the current block will be compensated using the motion vector of the block itself, plus the motion vectors of the blocks to the left of, above, and above left of the current block. Each vector provides a prediction for the pel, and those four predictions are weighted according to the spatial position of the pel within the block. For example, as the spatial position of the pel gets closer to the left border of the block, a higher weight is given to the prediction provided by the motion vector of the block to the left.

Orchard *et al.* [102, 103] used this view to formulate OMC as a linear estimator of the form

$$\hat{f}_t(\mathbf{s}) = \sum_{\mathbf{d}_n \in \mathcal{N}(\mathbf{s})} w_n(\mathbf{s})\, f_{t-\Delta t}(\mathbf{s} - \mathbf{d}_n), \tag{4.32}$$

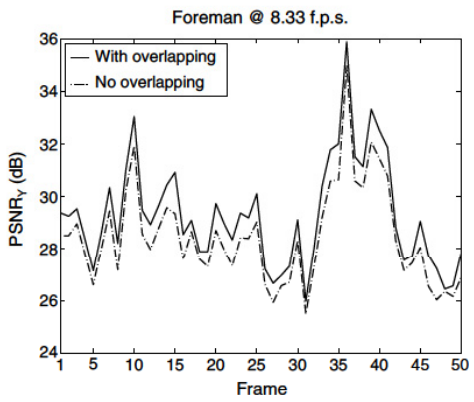
where $\mathcal{N}(\mathbf{s}) = \{\mathbf{d}_n(\mathbf{s})\}$ is the set of motion vectors used to compensate the pel at location $\mathbf{s}$ and $w_n(\mathbf{s})$ is the weight given to the prediction provided by vector $\mathbf{d}_n$. Using this formulation, they solve two optimization problems: overlapped-motion compensation and overlapped-motion estimation. Given the set of motion vectors $\mathcal{N}(\mathbf{s})$ estimated by the encoder, they propose a method for designing optimal windows, $w_n(\mathbf{s})$, to be used at the decoder for motion compensation. Also, given a fixed window that will be used at the decoder, they propose a method for finding the optimal set of motion vectors at the encoder. Note that the latter problem is much more complex than the BMA, since in this case the estimated motion vectors are interdependent. For this reason, their proposed method is based on an iterative procedure. A number of methods have been proposed to alleviate this complexity, e.g., Ref. 104.

As a linear estimator of intensities, OMC belongs to a more general set of motion compensation methods called *multihypothesis motion compensation*. Another member in this set is bidirectional motion compensation. The theoretical motivations for such methods were presented by Sullivan in 1993 [105]. Recently, Girod [106] analyzed the rate-distortion efficiency of such methods and provided performance bounds and comparisons with single-hypothesis motion compensation (e.g., the BMA).

Figure 4.9 compares the performance of OMC to that of the BMA when applied to the FOREMAN sequence. In the case of OMC, the same BMA motion vectors were used for compensation (i.e., the motion vectors were not optimized for overlapped compensation). Each motion vector was used to copy a 32×32 block from the reference frame and center it around the current 16×16 block in the current frame. Each copied block was weighted by a bilinear window function defined as [103]

$$w(x, y) = w_x \cdot w_y, \quad \text{where } w_z = \begin{cases} \frac{1}{16}(z + \frac{1}{2}) & \text{for } z = 0, \ldots, 15, \\ w_{31-z} & \text{for } z = 16, \ldots, 31. \end{cases} \tag{4.33}$$

Border blocks were handled by assuming "phantom" blocks outside the frame boundary with motion vectors equal to those of the border blocks. Despite the fact that the estimated vectors, the window shape, and the overlapping weights were not optimized for overlapped compensation, OMC provided better objective (Figure 4.9(a)) and subjective (Figures 4.9(b)–4.9(d)) quality compared to the BMA. In particular, the annoying blocking artefacts have clearly been reduced.



(a) Prediction quality



(b) Original 15th frame at 8.33 f.p.s



(c) Compensated using BMA (29.35 dB)



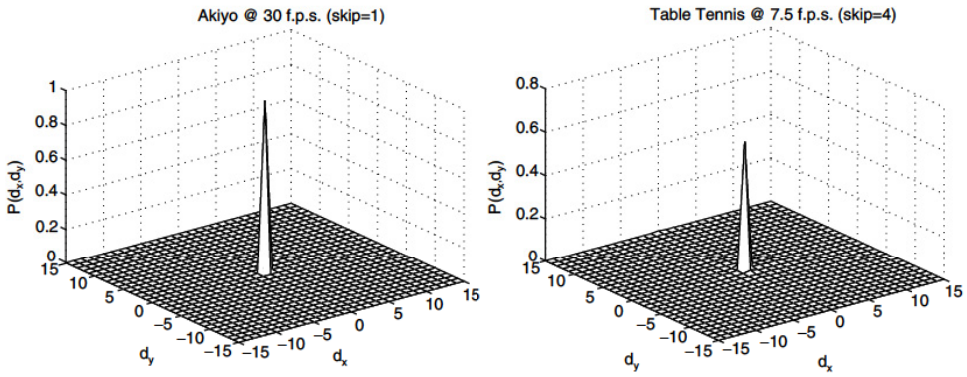
(d) Compensated using OMC (30.93 dB)

Figure 4.9: Comparison between OMC and BMA

4.6.7 Properties of Block-Motion Fields and Error Surfaces

This subsection presents some basic properties of the BMME algorithm when applied to typical video sequences. These properties will be utilized and referenced in subsequent chapters of the book. All illustrations in this subsection were generated using a full-pel full-search block-matching algorithm with 16×16 blocks, ± 15 pels maximum displacement, restricted motion vectors, SAD as the BDM, and original reference frames.

Property 4.6.7.1 *The distribution of the block motion field is center-biased.* This means that smaller displacements are more probable and the motion vector $(0,0)$ has the highest probability of occurrence. In other words, most blocks are stationary or quasi-stationary. This property is illustrated in Figure 4.10(a) for AKIYO at 30 frames/s (frame skip of 1). The property also holds true for



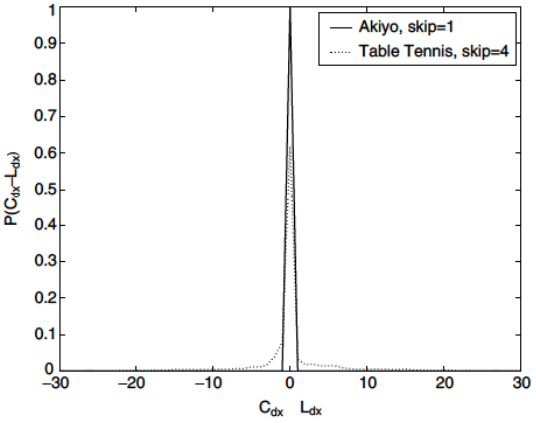
(a) AKIYO at 30 frames/s (skip=1) (b) TABLE TENNIS at 7.5 frames/s (skip=4)

Figure 4.10: Center-biased distribution of block-motion field

Foreman at 25 frames/s

$\rho_x = 0.56$ $\rho_y = 0.33$	$\rho_x = 0.69$ $\rho_y = 0.49$	$\rho_x = 0.64$ $\rho_y = 0.46$
$\rho_x = 0.66$ $\rho_y = 0.48$	current block	$\rho_x = 0.72$ $\rho_y = 0.63$
$\rho_x = 0.64$ $\rho_y = 0.43$	$\rho_x = 0.76$ $\rho_y = 0.61$	$\rho_x = 0.68$ $\rho_y = 0.56$

(a) Correlation coefficients betwee the motion vector of a block and its eight neighboring blocks



(b) Distribution of the diffference between the horizontal component of the current vector and that of its left neighbor

Figure 4.11: Highly correlated block-motion fields

sequences with higher motion content and at lower frame rates, as illustrated in Figure 4.10(b) for TABLE TENNIS at 7.5 frames/s (frame skip of 4).

Property 4.6.7.2 *The block motion field is smooth and varies slowly*. In other words, there is high correlation between the motion vectors of adjacent blocks. Thus, it is very common to find neighboring blocks with identical or nearly identical motion vectors. This is evident in Figure 4.11(a), which shows the correlation coefficients between the motion vector of a block and its eight

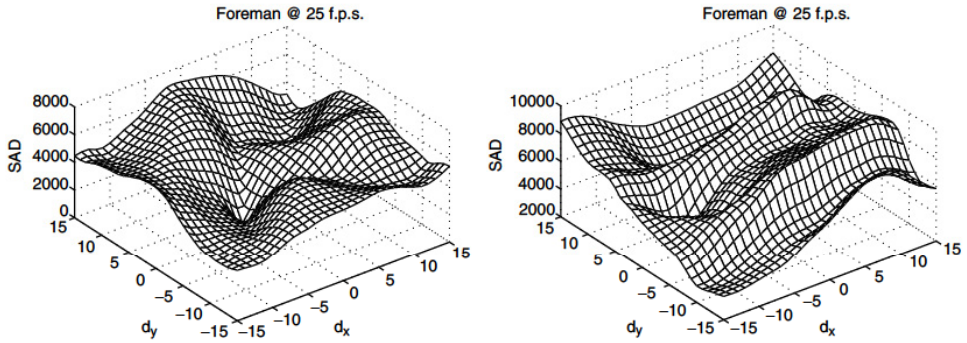


Figure 4.12: Sample multimodal error surfaces

neighboring blocks in FOREMAN at 25 frames/s. This is also illustrated in Figure 4.11(b), which shows the distribution of the difference between the horizontal component of the current vector (C_{d_x}) and that of its left neighbor (L_{d_x}). The bias of this distribution toward the zero difference clearly indicates high correlation, and this holds true for both AKIYO at 30 frames/s and TABLE TENNIS at 7.5 frames/s.

Property 4.6.7.3 *The error surface is usually multimodal.* In most cases, the error surface will contain one or more local minima, as illustrated in Figure 4.12. This can be due to a number of reasons, for example, the ambiguity problem, the accuracy problem, and the textured (periodical) local frame content.

Property 4.6.7.4 *The value of the global minimum of an error surface can change according to many factors, such as the frame skip, the motion content, and the block content.* For example, Figure 4.12 shows the error surface of two blocks from the same frame. The value of the global minimum of the surface in Figure 4.12(a) is 614, whereas that of the surface in Figure 4.12(b) is 3154.

4.7 A Comparative Study

This section presents the results of a comparative study of the motion estimation methods discussed in Sections 4.3–4.6. The main aim of this study is to answer the following question: *What is the best motion estimation algorithm for video coding?* In this study, the following algorithms were implemented:

DFA This is an implementation of the differential method of Cafforio and Rocca as given by Equation (4.12). In this case, the moving area, $\mathcal{A}$, was set to a block of 16×16 pels.

PRA This is an implementation of the *pel-recursive algorithm* of Netravali and Robbins as given by Equation (4.20). In this case, the motion vector of the previous pel in the line was taken as the initial motion estimate, $\hat{\mathbf{d}}^i$, of the current pel, the update step size was set to $\varepsilon = 1/1024$, the update term was calculated and averaged over an area of 3×3 pels centered around the current pel, and five iterations were performed per pel.

PCA This is an implementation of the phase correlation method as given by Equations (4.24) and (4.25). In this case, a window of 32×32 pels centered around the current 16×16 block was used to generate the phase correlation surface. The three most dominant peaks in this surface were detected and the corresponding motion displacements were unwrapped using Equation (4.27). The three candidate displacements were then tested using the SAD between the current block and the candidate displaced block in the reference frame. The candidate displacement with the lowest SAD was chosen as the motion vector of the current block.

BMA This is an implementation of a full-search block-matching algorithm. In this case, the block size was 16×16 pels and the matching criterion was the SAD.

In each case, the maximum allowed motion displacement was set to ± 15 pels in each direction and the motion vectors were allowed to point outside the reference frame (i.e., unrestricted motion vectors). To provide a fair comparison and to ease motion vector coding, all displacements were estimated with half-pel accuracy. In DFA and PRA this was achieved by rounding the subpel accurate motion estimates to the nearest half-pel accurate motion vectors. In PCA and BMA this was achieved using a refinement stage that examined the eight nearest half-pel estimates centered around the full-pel motion estimate. Bilinear interpolation was used to obtain intensity values at subpel locations of the reference frame. To mask the effect of the temporal propagation of prediction errors, motion was estimated and compensated using original reference frames. For comparison purposes, motion vectors were coded using the median predictor and the VLC table of the H.263 standard. The DFD signal was also transform encoded according to the H.263 standard and a quantization parameter of $QP = 10$. All quoted results refer to the luma components of sequences. No chroma encoding was performed.

Care should be taken when interpreting the results of this study. Different simulation parameters will lead to different results. For example, at the expense

of a higher computational complexity, the performance of the PRA can be improved by increasing the number of iterations. This is also true when examining more peaks for the PCA.

Figure 4.13 compares the prediction quality of the four algorithms when applied to the three test sequences AKIYO, FOREMAN, and TABLE TENNIS, at different frame skips (and, consequently, different frame rates).

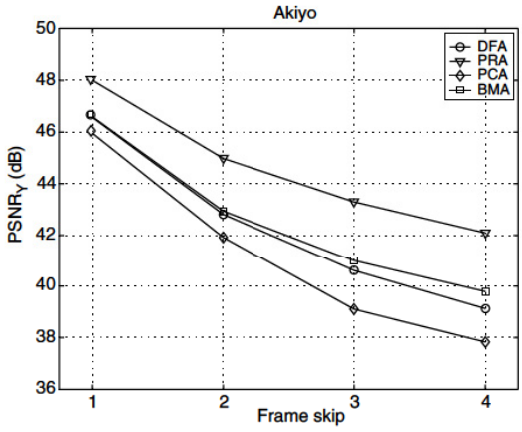
As expected, the DFA performs well for sequences with a low amount of movement (AKIYO) and at low frame skips (i.e., high frame rates). For sequences with a higher amount of movement (FOREMAN and TABLE TENNIS) and also at high frame skips, the motion vectors become longer, the quality of the Taylor series approximation becomes poor, and the performance of the DFA deteriorates.

Due to its dense motion field, the PRA has a superior performance for AKIYO and a very competitive performance for FOREMAN and TABLE TENNIS. The relative drop in performance for high-motion sequences and at high frame skips may be due to a number of reasons. With longer motion vectors, there is more possibility that the algorithm will be trapped in a local minimum before reaching the global minimum. Also, the maximum number of iterations may not be sufficient to reach the global minimum. However, increasing the number of iterations will increase the complexity of the algorithm.

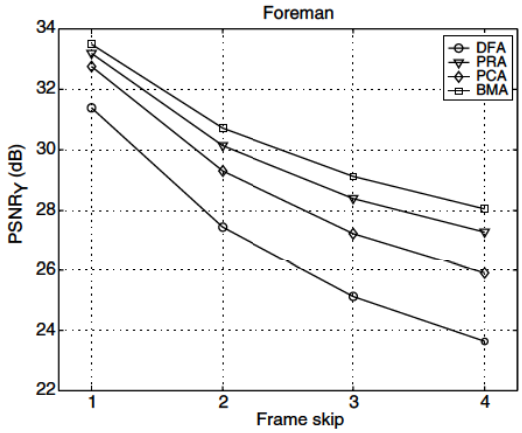
In general, the performance of the PCA is somewhere in between that of the DFA and PRA. The poor performance for AKIYO may be due to the spurious peaks produced by the boundary and spectral leakage effects. Such effects may be reduced by applying a weighting function to smooth the phase correlation surface.

The best overall performance is provided by the BMA. It performs well regardless of the sequence type and the frame skip. In fact, for sequences with a high amount of movement (FOREMAN and TABLE TENNIS), the BMA shows superior performance.

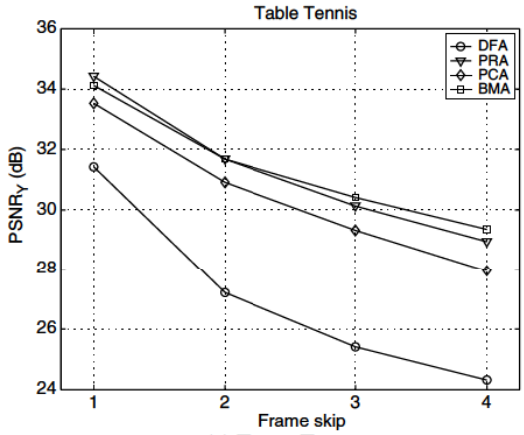
It is interesting at this point to concentrate on the PRA and BMA, for two reasons. First, they achieved the best prediction quality performance in the comparison. Second, they represent two different approaches to motion estimation (pel-based and block-based, respectively). Figure 4.14 compares the performance of the PRA and the BMA for the first 50 frames of the FOREMAN sequence at 25 frames/s. Two versions of the PRA are considered: PRA, which is the same algorithm described earlier, and PRA-C, which is an algorithm in which the update term is based on the *causal* part of an area of 5×5 pels centered around the current pel. Since PRA-C is based on causal data, no motion overhead needs to be transmitted for this method. Due to the high amount of motion in FOREMAN, the maximum number of iterations for both pel-recursive algorithms was increased to 10.



(a) AKIYO



(b) FOREMAN



(c) TABLE TENNIS

Figure 4.13: Prediction quality of different motion estimation algorithms

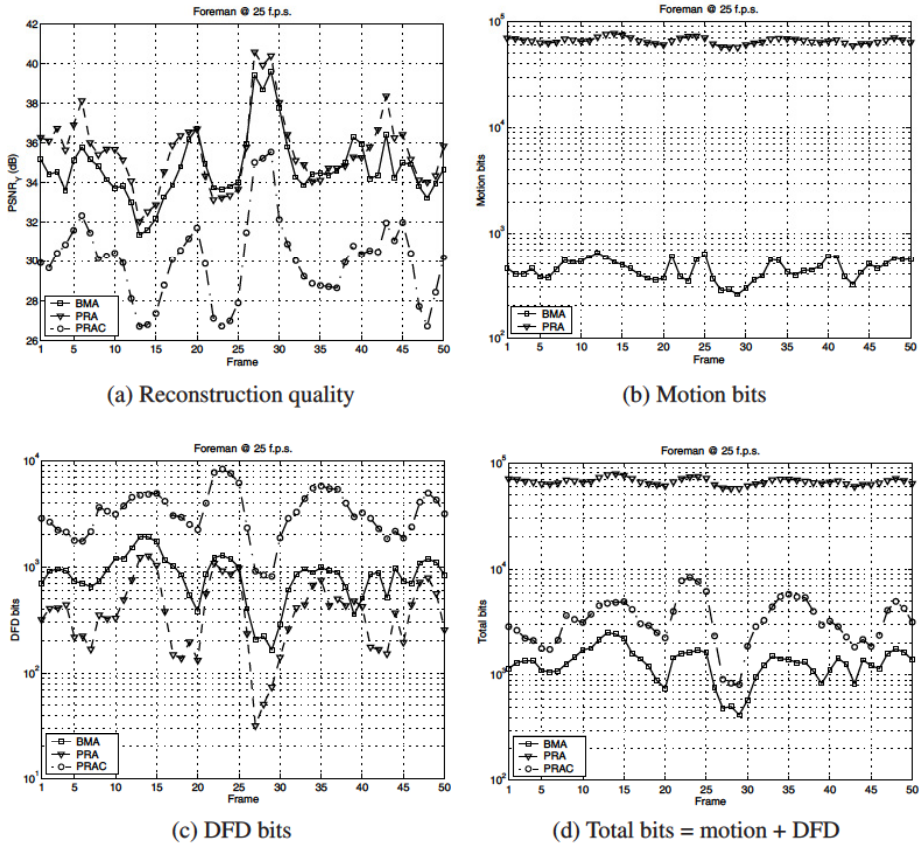


Figure 4.14: Comparison between BMA and PRA motion estimation algorithms

The aim of motion estimation for video coding is to simultaneously minimize the bit rate corresponding both to the motion parameters (motion bits) and to the prediction error signal (DFD bits). As illustrated in Figure 4.14, the three algorithms represent three different tradeoffs between prediction quality and motion overhead. Due to its dense motion field, the PRA has the best prediction quality and, consequently, the least DFD bits. This is, however, at the expense of a prohibitive motion overhead, which leads to a very high total bit rate. The causal implementation of the PRA, PRA-C, clearly restricts the method and significantly reduces its prediction quality. Thus, PRA-C removes the motion overhead at the expense of an increase in DFD bits. In addition, this causal implementation increases the complexity of the decoder. The best tradeoff is achieved by the BMA. It uses a block-based approach to reduce the motion overhead while still maintaining a very good prediction quality. This explains the popularity of this approach and its inclusion in video coding standards.

4.8 Efficiency of Block Matching at Very Low Bit Rates

The incorporation of motion estimation and compensation into a video codec involves extra computational complexity. This extra complexity must, therefore, be justified on the basis of an enhanced coding efficiency. This is very important for very-low-bit-rate applications and, in particular, for applications like mobile video communication, where battery time and processing power are scarce resources.

Very low bit rates are usually associated with high frame skips. As the frame skip increases, the temporal correlation between consecutive frames decreases. This will obviously decrease the efficiency of motion estimation, as can be seen in Figure 4.13. This poses a very important question: Is the use of motion estimation at such bit rates justifiable? Or put in another way, Is the use of less complex coding methods, like frame differencing and intraframe coding, sufficient at those bit rates?

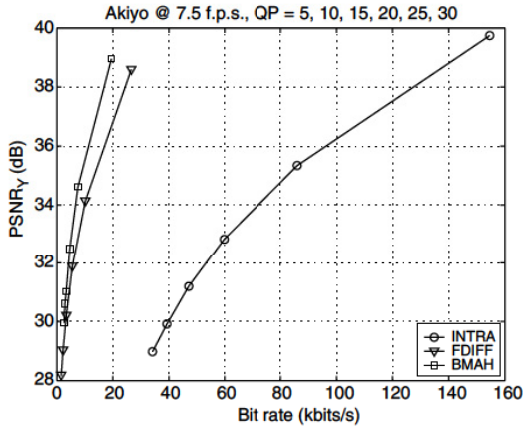
This study investigates the efficiency of block-matching motion estimation at very low bit rates. Three algorithms were implemented:

BMA-H This is a half-pel full-search BMA with 16×16 blocks, ± 15 pels maximum displacement, restricted motion vectors, and SAD as the matching criterion. Half-pel accuracy is achieved using a refinement stage around the full-search full-pel motion vectors. Bilinear interpolation is used to obtain intensity values at subpel locations of the reference frame.

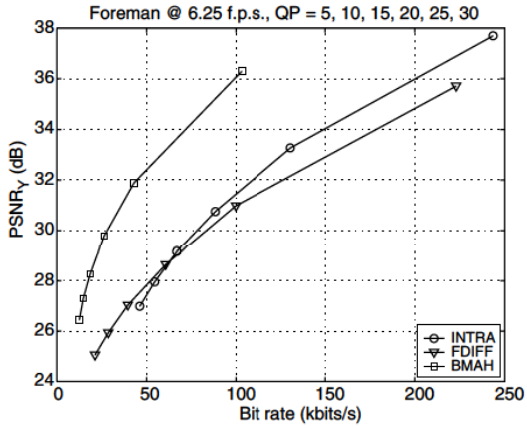
FDIFF This is a frame differencing algorithm. This means that no motion estimation is performed and the motion vectors are always assumed to be $(0,0)$. Note that this algorithm has no motion overhead and the total frame bits are equal to the DFD bits.

INTRA This is a DCT-based intraframe coding algorithm.

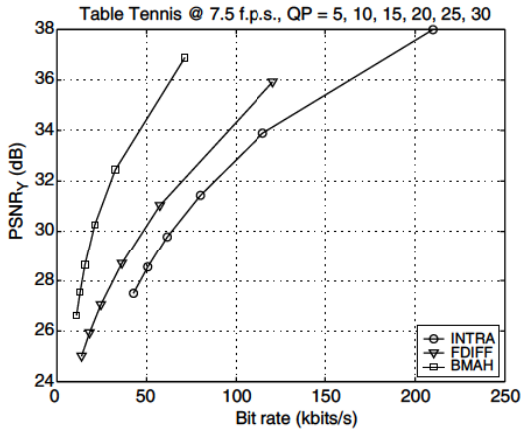
In each algorithm, motion was estimated and compensated using reconstructed reference frames. Motion vectors were coded using the median predictor and the VLC table of the H.263 standard. Both, the DFD signal (in the case of BMA-H and FDIFF) and the frame signal (in the case of INTRA) were transform encoded according to the H.263 standard. To simulate a very-low-bit-rate environment, the frame skip was set to 4 (this corresponds to 7.5 frames/s for AKIYO and TABLE TENNIS and to 6.25 frames/s for FOREMAN). To generate a range of bit rates, the quantization parameter QP was varied over the range 5–30 in steps of 5. This means that each algorithm was used to encode a given sequence six times. Each time, QP was held constant over the whole sequence (i.e., no rate control was used). The first frame of a sequence was always INTRA coded, regardless of the encoding algorithm, and the



(a) AKIYO



(b) FOREMAN



(c) TABLE TENNIS

Figure 4.15: Efficiency of block-matching motion estimation at very low bit rates

resulting bits were included in the bit-rate calculations. All quoted results refer to the luma components of sequences. Figure 4.15 compares the performance of the three algorithms when applied to the three test sequences.

In general, both interframe coding algorithms (FDIFF and BMA-H) outperform the intraframe coding algorithm (INTRA). Thus, even at very low bit rates, high frame skips, or low-motion sequences, the temporal correlation between video frames is still high enough to justify interframe coding.

Comparing the two interframe coding algorithms, it is immediately evident that the BMA-H algorithm outperforms the FDIFF algorithm at all bit rates and for all sequences. Note, however, that at extremely low bit rates, and in particular for the low-motion AKIYO sequence, the efficiency of the BMA-H algorithm starts to drop and approaches that of the simpler FDIFF algorithm. But even with this drop in performance, the use of BMA-H is still justifiable. For example, with AKIYO and at a bit rate as low as 3 kbits/s, the BMA-H algorithm still outperforms the FDIFF algorithm by about 1 dB.

4.9 Discussion

Motion estimation is an important process in a wide range of applications. Different applications have different requirements and may, therefore, employ different motion estimation techniques.

In video coding, the determination of the *true* motion is not the intrinsic goal. The aim is rather to simultaneously minimize the bit rate corresponding both to the motion parameters (motion bits) and to the prediction error signal (DFD bits). This is not an easy task, since the minimization of one quantity usually leads to maximizing the other. Thus, a suitable tradeoff is usually sought. In this chapter, four motion estimation methods were compared. The four methods are the differential, pel-recursive, phase-correlation, and block-matching motion estimation methods. It was found that block-matching motion estimation provides the best tradeoff. It uses a block-based approach to reduce the motion overhead while still maintaining a very good prediction quality (and consequently a small number of DFD bits). This explains the popularity of this approach and its inclusion in video coding standards.

The chapter also investigated the efficiency of motion estimation at very low bit rates. It was found that the prediction quality of motion estimation starts to drop at very low bit rates, in particular, for low-motion sequences, and approaches that of simpler techniques, like frame differencing and intraframe coding. Despite this drop in prediction quality, it was found that the use of motion estimation is still justifiable at those bit rates.

Chapter 5

Warping-Based Motion Estimation Techniques

5.1 Overview

As already discussed, one way to achieve higher coding efficiency is to improve the performance of the motion estimation and compensation processes. This can be done by using advanced motion estimation and compensation techniques. This chapter concentrates on an advanced technique called *warping-based motion estimation*. Since the early 1990s, this technique has attracted attention in the video coding community as an alternative to (or rather as a generalization of) conventional block-matching methods.

Section 5.2 reviews warping-based motion estimation techniques. Various aspects of such techniques, like the shape of patches, the type of meshes, the spatial transformation, the continuity of the motion field, the direction of node tracking, the node tracking algorithm, the motion compensation method, and the transmitted motion overhead, are considered and compared. Section 5.3 compares the performance of warping-based methods to that of block-matching methods. In particular, the section investigates the efficiency of warping-based methods at very low bit rates. The chapter concludes with a discussion in Section 5.4.

5.2 Warping-Based Methods: A Review

Motion estimation (ME) can be defined as a process that divides the current frame, f_c, into regions and that estimates for each region a set of *motion parameters*, $\{a_i\}$, according to a motion model. The *motion compensation* (MC) process then uses the estimated motion parameters and the motion model to *synthesize* a prediction, $\hat{f}_c$, of the current frame from a reference frame, f_r.

This synthesis process can be formulated as follows:

$$\hat{f}_c(x, y) = f_r(u, v), \tag{5.1}$$

where (x, y) are the spatial coordinates in the current frame (or its prediction) and (u, v) are the spatial coordinates in the reference frame. This equation indicates that the MC process applies a *geometric transformation* that maps one coordinate system onto another. This is defined by means of the *spatial transformation* functions g_x and g_y:

$$\begin{aligned} u &= g_x(x, y), \\ v &= g_y(x, y). \end{aligned} \tag{5.2}$$

This spatial transformation is also referred to as *texture mapping* or *image warping* [107].

As already discussed, the BMA method relies on a uniform translational motion model. Thus, the transformation functions of this method are given by

$$\begin{aligned} u &= g_x(x, y) = x + a_1 = x + d_x, \\ v &= g_y(x, y) = y + a_2 = y + d_y. \end{aligned} \tag{5.3}$$

In practice, however, a block can contain multiple moving objects, and the motion is usually more complex and can contain translation, rotation, shear, expansion, and other deformation components. In such cases, the simple uniform translational model will fail, and this will usually appear as artefacts, e.g., blockiness, in the motion-compensated prediction. Higher-order motion models can be used to overcome such problems. Examples of such models are the affine, bilinear, and perspective spatial transformations given by Equations (5.4), (5.5), and (5.6), respectively:

Affine:

$$\begin{aligned} u &= g_x(x, y) = a_1 x + a_2 y + a_3, \\ v &= g_y(x, y) = a_4 x + a_5 y + a_6. \end{aligned} \tag{5.4}$$

Bilinear:

$$\begin{aligned} u &= g_x(x, y) = a_1 xy + a_2 x + a_3 y + a_4, \\ v &= g_y(x, y) = a_5 xy + a_6 x + a_7 y + a_8. \end{aligned} \tag{5.5}$$

Perspective:

$$u = g_x(x, y) = \frac{a_1 x + a_2 y + a_3}{a_7 x + a_8 y + 1},$$
$$v = g_y(x, y) = \frac{a_4 x + a_5 y + a_6}{a_7 x + a_8 y + 1}. \tag{5.6}$$

Motion estimation and compensation using higher-order models is usually performed using the following steps:

1. A 2-D *mesh* is used to divide the current frame into nonoverlapping polygonal patches (or elements). The points shared by the vertices of the patches are referred to as *grid* or *node points*.
2. The motion of each node is estimated. This will map each node in the current frame to a corresponding node in the reference frame. In effect, this will map each patch in the current frame to a corresponding patch in the reference frame.
3. For each patch in the current frame, the coordinates of its vertices and those of the matching patch in the reference frame are used to find the motion parameters $\{a_i\}$ of the underlying motion model.
4. During motion compensation, the estimated motion parameters $\{a_i\}$ are substituted in the appropriate spatial transformation, Equations (5.4)–(5.6), to *warp* the patch in the reference frame to provide a prediction for the corresponding patch in the current frame.

An example of this process is illustrated in Figure 5.1. In this figure the current frame is divided into square patches. This forms a uniform mesh. During motion estimation, node points A, B, C, and D in the current frame are mapped to node points A′, B′, C′, and D′ in the reference frame. During motion compensation, the deformed patch A′B′C′D′ is warped to provide a prediction for the square patch ABCD.

It should be pointed out that there is a lack of consistency in the literature when referring to this type of motion estimation and compensation methods. Examples of the numerous names employed are control grid interpolation [108, 109, 110], warping-based methods [111, 112, 113], spatial-transformation-based methods [114, 115, 116, 117], geometric-transformation-based methods [118], generalized motion estimation methods [119, 120], and mesh-based methods [121, 122, 123, 124, 125, 126, 127].

When designing a warping-based technique, several aspects of the method need to be considered and defined, as discussed in the following subsections.

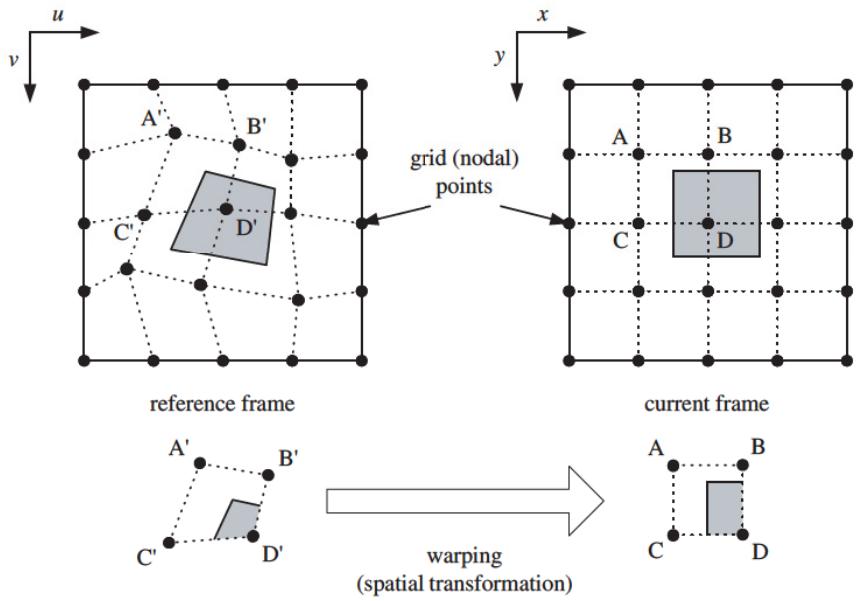


Figure-5.1:-Warping-based-motion-estimation-and-compensation-

5.2.1 Shape of Patches

The-most-widely-used-shapes-are-*triangles*-and-*quadrilaterals*.-Nakaya-and-Harashima-[114]-showed-that-equilateral-triangles-are-optimal,-in-the-prediction-quality-sense,-when-the-affine-transformation-is-used,-whereas-squares-are-optimal-when-the-bilinear-transformation-is-used.-Square-patches-are-sometimes-preferred-because-they-are-compatible-with-current-block-based-video-coding-methods-and-standards.-Triangular-patches-are-more-compatible-with-model-based-coding-methods,-where-wireframe-models-are-usually-defined-in-terms-of-triangles.-

5.2.2 Type of Mesh

The-mesh-structure-can-be-fixed-or-adaptive.-A-*fixed mesh*-is-one-that-is-built-according-to-a-predetermined-pattern,-e.g.,-a-regular-mesh-with-square-patches.-An-*adaptive mesh*,-on-the-other-hand,-is-one-that-is-adaptively-built-according-to-frame-contents-and-motion.-Adaptive-meshes-can-be-content-based-or-motion-based.-In-content-based-adaptive-meshes,-nodes-are-placed-to-fit-important-features-like-contours-and-edges-[111,-121].-In-motion-based-adaptive-meshes,-more-nodes-are-placed-in-moving-areas.-This-is-usually-achieved-using-a-hierarchical-(usually,-quad-tree)-mesh-structure-[109,-120,-

115,123]. Although adaptive meshes can improve prediction quality, they have the disadvantages of increased computational complexity (for the generation and adaptation processes) and increased overhead (to describe the structure of the mesh). The structure overhead can be removed by applying the adaptation process based on previous frames that are available at the decoder.

5.2.3 Spatial Transformation

As shown by Seferidis and Ghanbari [119], the perspective transform achieves the best prediction quality performance. However, the high computational complexity of this transformation limits its use in practice. The affine transformation is the least computationally complex, but it has the fewest degrees of freedom. The performance of the bilinear transformation is very close to that of the perspective transformation, with the advantage of reduced computational complexity. However, a study by Nakaya and Harashima [114] showed that the affine and bilinear transformations have almost the same performance when the patch shape is optimized (equilateral triangles and squares, respectively). In fact, the same study showed that the performance of the affine transformation can be superior as the number of nodes decreases.

5.2.4 Continuous Versus Discontinuous Methods

Adjacent patches in the current frame have common vertices between them. There are two main methods for estimating the motion of such common vertices. If the motion of common vertices is estimated independent from each other (i.e., common vertices are assigned different motion vectors), then this will result in a discontinuous motion field with discontinuities along the boundaries of the patches. This is known as the *discontinuous method*. The motion field in this case has similarities with that produced by the BMA. If, however, a restriction is applied such that common vertices have the same motion vector, then this will result in a continuous motion field and the method is known as the *continuous method*. The two methods are illustrated in Figure 5.2.

As pointed out by Ghanbari *et al.* [115], the discontinuous method is more flexible and can compensate for more general complex motion. However, as pointed out by Nakaya and Harashima [114], since discontinuities are allowed along the boundaries of patches, this method can suffer from blocking artefacts. Another disadvantage of the discontinuous method is that it generates more motion overhead (four motion vectors per patch) compared to the continuous method (about one motion vector per patch).

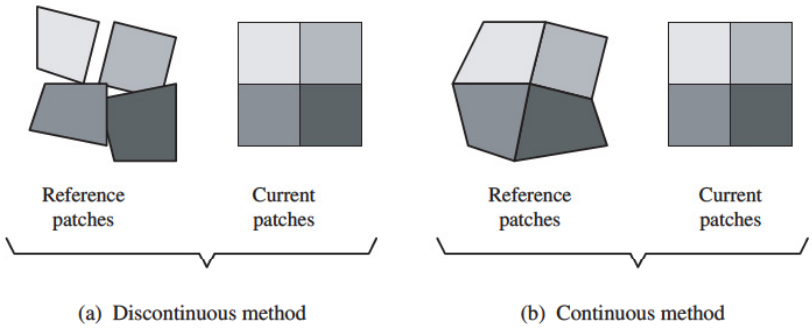


Figure 5.2: Continuous versus discontinuous warping-based methods

5.2.5 Backward Versus Forward Node Tracking

The process of estimating the motion of a grid or a node point is called *node tracking*. There are two types of node-tracking algorithms: backward and forward node tracking.

In *backward node tracking*, nodes are first placed on the current frame and then they are matched to points in the reference frame. During motion compensation, a pel (x, y) in the current patch is *copied from* a corresponding pel $(u, v) = (g_x(x, y), g_y(x, y))$ in the reference patch. Note that in this case, (x, y) is a sampling spatial position, whereas (u, v) may be a nonsampling spatial position. Interpolation, e.g., bilinear, can be used to obtain pel values at nonsampling positions of the reference frame. This process is repeated for all pels within the current patch. Since backward tracking starts with a mesh on the current frame (which is not available at the decoder), this technique is usually used in combination with a fixed mesh.

In *forward node tracking*, nodes are first placed on the reference frame and then matched to points in the current frame. During motion compensation, a pel (u, v) in the reference patch is *copied to* a corresponding pel $(x, y) = (g_x(u, v), g_y(u, v))$ in the current patch. Since, in this case, (x, y) may be a nonsampling spatial position, the compensated current patch will normally contain holes (i.e., noncompensated pels at sampling spatial positions). Techniques that can be used to recover pel values at sampling spatial positions from values at nonsampling spatial positions are discussed and compared by Sharaf and Marvasti in Ref. 116. Due to the use of such techniques, forward node tracking and compensation is computationally more complex than backward node tracking and compensation. Since forward node tracking starts with a mesh on the reference frame, this technique is usually used in combination with an adaptive mesh. Although the combination of forward tracking and adaptive meshes can provide some prediction quality improvement over the

combination of backward tracking and fixed meshes, the use of the former is not justified, due to the huge increase in computational complexity [116].

5.2.6 Node-Tracking Algorithm

A simple method to estimate the motion of a node is to use a BMA-type algorithm which minimizes the translational prediction error in a block centered around the node. Niewęgłowski *et al.* [111] use a modified BMA with a large block (21×21) centered around the node and a distortion measure designed to give more weight to pels closer to the node. To reduce complexity, the block is subsampled by a factor of 2:1 in both directions.

Although BMA-type algorithms are simple, they provide suboptimal performance. First, they assume that the motion of a node is independent of the motion of other nodes, and second, they assume that minimizing the translational prediction error minimizes the true prediction error. In practice, however, both assumptions are not true. A node is a common vertex between more than one patch. Consequently, the displacement of a node will affect all patches connected to it. For example, with quadrilateral patches, the displacement of a node affects the prediction quality within four patches connected to it. It follows that the choice of the motion vector of one node will affect the choice of the motion vectors of other nodes. In addition, the true prediction error is the error between the current frame and its warped prediction from the reference frame. This is not equal to the translational prediction error.

Brusewitz [128] uses a BMA-type algorithm to provide coarse approximations for nodal motion vectors. An iterative gradient-based approach that minimizes the true prediction error is then used to refine all nodal motion vectors simultaneously. The computational complexity of the method is extremely high. For example, if there are 100 nodes in the frame, the method requires the inversion of a 200×200 matrix.

To reduce complexity, Sullivan and Baker [108] estimate the motion of one node at a time. However, to take into account the interdependence between motion vectors, an iterative approach is employed. In each iteration, the nodes are processed sequentially. The motion vector of a node is estimated using a local search around the motion vector from the previous iteration while holding constant the motion vectors of its surrounding nodes. During the local search, the quality of a candidate motion vector is measured by calculating the distortion measure between all patches connected to the node and their warped predictions from the reference frame. The local search is applied to a node only if its motion vector, or the motion vector of at least one of its surrounding nodes, was changed in the previous iteration.

Nakaya and Harashima [114] use a *hexagonal matching algorithm* (HMA). The name is due to the use of triangular patches for which each node is a

common vertex between six patches forming a hexagon. The algorithm is almost identical to that of Sullivan and Baker (described earlier). In this case, however, a BMA-type algorithm is first used to provide a coarse estimate of the motion field, and the iterative approach is then used to refine this estimate. In addition to the exhaustive local search, they also propose a faster but suboptimal gradient-based local search. Similar gradient-based approaches have also been used by Wang *et al.* [123, 126] and Dudon *et al.* [124].

Altunbasak and Tekalp [125] first estimate a dense field of motion displacements. Then they use a least squares method to estimate the nodal motion vectors subject to the constraint of preserving the connectivity of the mesh. They show that the performance of this algorithm is comparable to that of HMA, with the advantage of reduced computational complexity.

When estimating a nodal motion vector, it is very important to ensure that the estimate does not cause any patch connected to the node to become degenerate (i.e., with obtuse angles and/or flipover nodes). To accomplish this, Wang *et al.* [123, 126] limit the search range to a diamond region defined by the four surrounding nodes, whereas Altunbasak and Tekalp [125] use a postprocessing stage where an invalid estimate is replaced by a valid estimate interpolated from surrounding nodal motion vectors.

All the foregoing algorithms assume a continuous motion field. Ghanbari *et al.* [119, 120, 115, 117] use quadrilateral patches with a discontinuous motion field. In this case, the four vertices of each regular patch in the current frame are displaced combinatorially (i.e., perturbed) to find the best-match deformed patch in the reference frame. The computational complexity of this algorithm is extremely high since there are $(2d_m + 1)^8$ possible deformed patches in the reference frame. In addition, each possible patch must first be warped to calculate the distortion measure. To reduce complexity, they propose to use a fast-search algorithm, e.g. Ref. 129.

5.2.7 Motion Compensation Method

Having obtained nodal motion vectors, there are two methods of performing motion compensation.

In the first method, for each patch in the current frame, the coordinates of its vertices and those of the matching patch in the reference frame are used to set up a number of simultaneous equations. This set is then solved for the motion parameters $\{a_i\}$ of the underlying motion model. For example, assume a mesh of quadrilateral patches and a bilinear motion model. If the spatial coordinates of the top-left, top-right, bottom-left, and bottom-right vertices of the patch in the current frame are (x_A, y_A), (x_B, y_B), (x_C, y_C), and (x_D, y_D), respectively, and the corresponding estimated motion vectors are $\mathbf{d}_A$, $\mathbf{d}_B$, $\mathbf{d}_C$, and $\mathbf{d}_D$, respectively, then the spatial coordinates of the matching vertices in

the reference frame are (u_A, v_A), (u_B, v_B), (u_C, v_C), and (u_D, v_D), respectively, where, e.g., $(u_A, v_A) = (x_A + d_{x_A}, y_A + d_{y_A})$. Using the bilinear model of Equation (5.5), the following set of simultaneous equations is obtained:

$$\begin{bmatrix} u_A & u_B & u_C & u_D \\ v_A & v_B & v_C & v_D \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ a_5 & a_6 & a_7 & a_8 \end{bmatrix} \cdot \begin{bmatrix} x_A y_A & x_B y_B & x_C y_C & x_D y_D \\ x_A & x_B & x_C & x_D \\ y_A & y_B & y_C & y_D \\ 1 & 1 & 1 & 1 \end{bmatrix}. \tag{5.7}$$

This set can easily be solved for the motion parameters $a_1, \ldots, a_8$. Having obtained the motion parameters of the current patch, each pel (x, y) in the patch is then compensated from a pel (u, v) in the reference patch, where (u, v) are obtained using Equation (5.5).

In the second method of motion compensation (commonly known as *control grid interpolation* (CGI) [108]), the motion vectors at the vertices of the current patch are interpolated to produce a dense motion field within the patch. For the same example just given, the motion vector $\mathbf{d}(x, y) = (d_x(x, y), d_y(x, y))$ at pel (x, y) of the current patch is obtained by bilinear interpolation of the four motion vectors at the vertices. Thus

$$\mathbf{d}(x, y) = (1 - x_n)(1 - y_n)\mathbf{d}_A + x_n(1 - y_n)\mathbf{d}_B + (1 - x_n)y_n\mathbf{d}_C + x_n y_n \mathbf{d}_D, \tag{5.8}$$

$$\text{where} \quad x_n = \frac{x - x_A}{x_B - x_A} \quad \text{and} \quad y_n = \frac{y - y_A}{y_C - y_A}. \tag{5.9}$$

Each pel (x, y) in the current patch can then be compensated from pel (u, v) in the reference patch, where $(u, v) = (x + d_x(x, y), y + d_y(x, y))$. It can be shown [110] that the two methods are equivalent.

5.2.8 Transmitted Motion Overhead

Two types of motion overhead can be transmitted: the motion parameters a_i of the patches and the motion vectors of the nodes. Motion vectors have a limited range and are usually evaluated to a finite accuracy (e.g., full- or half-pel accuracy), whereas motion parameters are not limited and are usually continuous in value. Thus, motion vectors are usually preferred because they are easier to encode and result in a more compact representation. In addition, motion vectors ensure compatibility with current video coding standards. One disadvantage in this case, however, is that the decoder is more complex, since it must use the received motion vectors to calculate the motion parameters

or to interpolate the motion field (as described in Section 5.2.7) before being able to perform motion compensation.

5.3 Efficiency of Warping-Based Methods at Very Low Bit Rates

This section investigates the performance of warping-based methods and compares it to that of block-matching methods. The main aim is to answer the following question: *Are there any gains for using higher-order motion models at very low bit rates?* In other words, this section assesses the suitability of warping-based methods for applications like mobile video communication.

Most results reported in the literature compare a warping-based algorithm to the basic block-matching algorithm. The authors feel that this is an unfair comparison for the following reasons:

1. As shown in Section 5.2.7, in warping-based compensation the motion vector used to compensate a pel in a given patch is interpolated from the nodal motion vectors at the vertices of the patch. Although the nodal motion vectors may be at full-pel accuracy, the resulting interpolated motion vector is at subpel accuracy. It is unfair to compare this subpel compensation to the full-pel compensation of the basic block-matching algorithm. A more fair comparison would be with a subpel (at least half-pel) block-matching algorithm.

2. Again, from Section 5.2.7, a warping-based method calculates one motion vector per pel. Thus, each pel within a patch is compensated individually. It is unfair to compare this to the basic block-matching algorithm, where the whole block is compensated using the same motion vector. A fairer comparison would be with overlapped motion compensation, where each pel within the block is compensated individually, as evident from Equation (4.32).

3. A warping-based method is much more computationally complex than the basic block-matching method (as is shown later). This increased complexity gives the warping-based method an unfair advantage over the basic block-matching method. To provide a fairer comparison, the basic block-matching method must be augmented by some advanced techniques (like subpel accuracy and overlapped compensation).

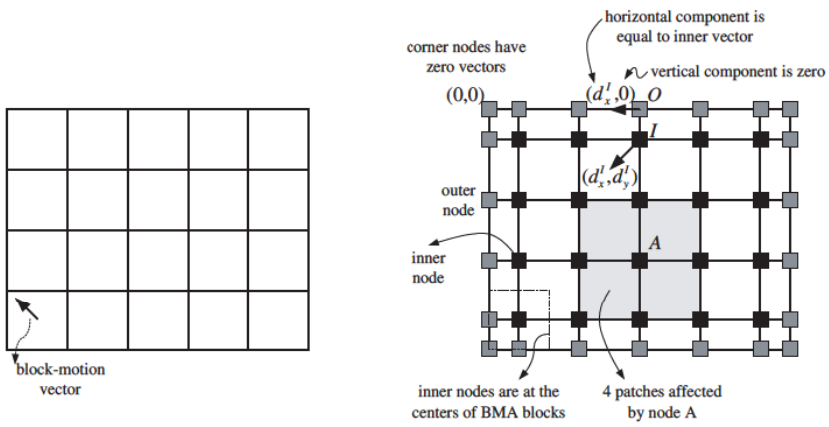
Thus, in this study, the following algorithms were implemented:

BMA This is a full-search full-pel block-matching algorithm with 16×16 blocks, restricted motion vectors, a maximum displacement of ± 15 pels, and SAD as the matching criterion.

BMA-HO This is the same as BMA but with half-pel accuracy and overlapped motion compensation. Half-pel accuracy was obtained using a refinement stage around the full-pel motion vector. Overlapping windows of 32 × 32 and a bilinear weighting function, Equation (4.33), were used for overlapped motion compensation. Border blocks were handled by assuming "phantom" blocks outside the frame boundary, with motion vectors equal to those of the border blocks.

WBA This is a *warping-based algorithm*. Node points were placed at the centers of 16 × 16 blocks in the current frame. This formed a regular fixed mesh with square patches. In order for the mesh to cover the whole frame area, node points were also placed on the borders.

Backward node tracking was used to map the current node points to their matches in the reference frame. A continuous method was used to produce a continuous motion field. To ensure that the number of transmitted motion vectors is the same as that of the BMA, no motion vectors were transmitted for the border node points. Instead, each border node was assigned the motion vector of the closest inner node. However, to ensure that the borders of the current frame were mapped to the borders of the reference frame, border nodes at the corners of the frame were assigned zero motion vectors, the vertical component of a top or a bottom border nodal vector was set to zero, and the horizontal component of a left or a right border nodal vector was set to zero. The mesh geometry and nodal motion vectors are illustrated in Figure 5.3.



(a) BMA blocks (b) WBA patches

Figure 5.3: BMA blocks and WBA patches

At the start of the node-tracking algorithm, the BMA described earlier was used to provide initial estimates of the inner nodal motion vectors. Those initial estimates were then refined using the iterative procedure of Sullivan and Baker [108]. In each iteration of this procedure, the nodes are processed sequentially, where the motion vector of a node is refined using a local search around the motion vector from the previous iteration while holding constant the motion vectors of its surrounding eight nodes. During this local search, the quality of a candidate motion vector is measured by calculating the distortion measure between all four patches connected to the node and their warped predictions from the reference frame. The local search is applied to a node only if its motion vector, or the motion vector of at least one of its surrounding nodes, was changed in the previous iteration. The local search used here examines the eight nearest candidate displacements centered around the displacement from the previous iteration. For each frame, 10 iterations were used to refine the nodal motion vectors.

During motion estimation and compensation, the bilinear spatial transformation is employed. This is implemented in the CGI [108] form (described in Section 5.2.7), where the motion vector used to compensate a pel within a patch is bilinearly interpolated, Equation (5.8), from the four nodal motion vectors at the vertices of the patch.

In BMA-HO and WBA algorithms, bilinear interpolation was used to obtain intensity values at subpel locations of the reference frame. In each algorithm, motion was estimated and compensated using original reference frames. Motion vectors were coded using the median predictor and the VLC table of the H.263 standard. The DFD signal was also transform encoded according to the H.263 standard and a quantization parameter of QP = 10. All quoted results refer to the luma components of sequences.

Table 5.1 compares the objective prediction quality of the preceding three algorithms when applied to the three test sequences with a frame skip of 3. The WBA outperforms the basic BMA by about 0.16–1.57 dB, depending on the sequence. However, the WBA fails to outperform the advanced BMA-HO

Table 5.1: Comparison between BMA and WBA in terms of objective prediction quality

	Average PSNR (dB) with a frame skip of 3		
	AKIYO	FOREMAN	TABLE TENNIS
BMA	39.88	27.81	29.06
WBA	41.45	29.09	29.22
BMA-HO	41.77	29.51	29.87

algorithm. In fact, the BMA-HO algorithm outperforms the WBA by about 0.32–0.65 dB.

Figure 5.4 compares the subjective prediction quality of the 45th frame of the 8.33-frames/s FOREMAN sequence when compensated using the preceding three algorithms. This figure shows that BMA-HO and WBA have approximately the same subjective quality and that both outperform the BMA. More importantly, this figure clearly shows the type of artefacts associated with each algorithm. The BMA suffers from the annoying blocking artefacts. Those artefacts are reduced by both the BMA-HO and the WBA algorithms. However, the BMA-HO algorithm has a low-pass filtering effect that smoothes sharp edges. This is due to the averaging (weighting) process during overlapped motion compensation. This effect is very clear at the edges of the helmet. The WBA, on the other hand, can suffer from warping artefacts. This is very clear at the top of the helmet, where part of the helmet was stretched to compensate



(a) Original 45th frame of FOREMAN at 8.33 f.p.s



(b) Compensated using BMA (28.06 dB)



(c) Compensated using BMA-HO (29.59 dB)



(d) Compensated using WBA (29.01 dB)

Figure 5.4: Comparison between BMA and WBA in terms of subjective prediction quality

uncovered-background.-In-fact,-poor-compensation-of-covered-and-uncovered-objects-is-one-of-the-main-disadvantages-of-the-continuous-warping-based-method.-In-particular,-the-method-performs-poorly-whenever-there-are-objects-disappearing-from-the-scene-because-it-can-deform-objects-but-cannot-easily-remove-them-completely-[111].-

Another-obvious-disadvantage-of-the-continuous-warping-based-method-is-the-lack-of-motion-field-segmentation.-A-number-of-methods-have-been-proposed-to-overcome-this-problem.-For-example,-Nieweglowski-and-Haavisto-[110]-use-adaptive-motion-field-interpolation-to-introduce-discontinuities-within-the-nodal-motion-field.-Adaptivity-is-achieved-by-switching-between-bilinear-interpolation-and-nearest-neighbor-interpolation-of-the-nodal-vectors-at-the-vertices-of-a-patch.-The-latter-interpolation-method-effectively-splits-the-motion-field-within-the-patch-into-four-quadrants.-A-similar-effect-can-be-achieved-by-using-a-hierarchical-(e.g.,-quad-tree)-motion-based-adaptive-mesh-[109,-120,-115,-123].-

It-is-interesting-at-this-point-to-compare-the-computational-complexity-of-the-preceding-three-algorithms.-Table-5.2-compares-the-complexity-of-the-three-algorithms-in-terms-of-encoding-time-per-frame.-The-results-were-obtained-using-the-profiler-of-the-Visual-C++-5.0 compiler-run-on-a-PC-with-Pentium-100-MHz-processor,-64-MB-of-RAM,-and-a-Windows-98-operating-system.-The-results-were-averaged-over-10 runs,-where-each-run-was-used-to-encode-the-8.33-frames/s FOREMAN-sequence.-Care-should-be-taken-when-interpreting-the-results-as-they-depend-heavily-on-the-implementation-and-the-hardware-platform.-

The-BMA-requires-about-2.16-seconds/frame.-Most-of-this-time-(about-1.76-seconds)-is-consumed-by-the-full-pel-full-search-block-matching-motion-estimation-process.-

The-BMA-HO-algorithm-requires-about-3.56-seconds/frame.-This-increase-of-about-1.4-seconds-over-the-BMA-is-due-mainly-to-two-reasons.-The-half-pel-refinement-stage-and-the-associated-bilinear-interpolation-process-increase-the-motion-estimation-time-by-about-0.98-seconds.-In-addition,-the-overlapping-process-increases-the-motion-compensation-time-by-about-0.42-seconds.-

Table-5.2:-Comparison-between-BMA-and-WBA-in-terms-of-computational-complexity-

	CPU-time-(in-seconds)-per-frame-when-encoding-FOREMAN-at-8.33-f.p.s-		
	BMA-	BMA-HO-	WBA-
BMA-motion-estimation-	1.76-	2.74-	1.86-
WBA-iterative-refinement-	0.00-	0.00-	116.00-
Motion-compensation-	0.01-	0.43-	0.60-
Others-	0.39-	0.38-	0.37-
Total-	**2.16**	**3.56**	**118.83**

The WBA requires about 118.83 seconds/frame. This is a huge increase over both the BMA and the BMA-HO algorithms. This increase is due mainly to the iterative procedure used to refine the initial nodal vector estimates. Remember that in each iteration, for a single node to be refined, spatial transformation and bilinear interpolation have to be used to compensate the four patches connected to the node. There are a number of methods that can be used to alleviate this complexity. Examples are the use of fewer iterations per frame, the use of a line-scanning[1] technique to perform the spatial transformation, the use of a simpler interpolation method (e.g., nearest neighbor) or the use of a noniterative motion estimation algorithm, e.g. Ref. 130. Most of these methods, however, reduce the computational complexity at the expense of a reduced prediction quality.

5.4 Discussion

Block matching methods have always been criticized because of their simple uniform translational model. The argument against this model is that, in practice, a block can contain multiple moving objects and the motion is usually more complex than simple translation. The shortcomings of this model may appear as poor prediction quality for objects with nontranslational motion and also as blocking artefacts within motion-compensated frames. Warping-based methods employing higher-order motion models have been proposed in the literature as alternatives to block-matching methods. This chapter investigated the performance of warping-based methods and compared it to that of block-matching methods. The results of this comparison have shown that despite their improvements over basic block-matching methods, the use of warping-based methods in applications like mobile video communication may not be justifiable, due to the huge increase in computational complexity. In fact, similar (if not better) improvements can be obtained, at a fraction of the complexity, by simply augmenting basic block-matching methods with advanced techniques like subpel accuracy and overlapped motion compensation. One can argue that warping-based methods can also benefit from subpel accuracy and overlapped motion compensation, as shown in Refs. 113 and 117, but again this will further increase complexity. In addition to their high computational complexity, warping-based methods can suffer from warping artefacts,

[1] Once the motion vector of a pel (x, y) within a patch is interpolated from the four nodal vectors at the vertices of the patch, it can be shown that the motion vectors of the next pel in the line $(x + 1, y)$ and the next pel in the column $(x, y + 1)$ can be obtained by adding a simple update term. This is known as line scanning [107].

poor-compensation-of-covered/uncovered-background,-and-lack-of-motion-field-segmentation.-Reducing-the-complexity-of-warping-based-methods-and-including-them-in-a-hybrid-WBA/BMA-video-codec-are-two-possible-areas-of-further-research.-

Chapter 6

Multiple-Reference Motion Estimation Techniques

6.1 Overview

To achieve high coding efficiency, Chapter 5 investigated an advanced motion estimation technique called warping-based motion estimation. This chapter considers another advanced technique, called *multiple-reference motion estimation*.

In *multiple-reference motion-compensated prediction* (MR-MCP), motion estimation and compensation are extended to utilize *more than one* reference frame. The reference frames are assembled in a *multiframe memory* (or buffer) that is maintained simultaneously at encoder and decoder. In this case, in addition to the spatial displacements, a motion vector is extended to also include a temporal displacement.

This chapter investigates the prediction gains achieved by MR-MCP. Particular emphasis is given to coding efficiency at very low bit rates. More precisely, the chapter attempts to answer the following question: *Is the use of additional bit rate to transmit the extra temporal displacement justifiable in terms of an improved rate-distortion performance?* The chapter also examines the properties of the multiple-reference block-motion field and compares them to those of the single-reference case.

The rest of the chapter is organized as follows. Section 6.2 briefly reviews multiple-reference motion estimation techniques. Section 6.3 concentrates on the long-term memory multiple-reference motion estimation technique. The section starts by examining the properties of multiple-reference block-motion fields and compares them to those of single-reference fields. It then investigates the prediction gains and the efficiency of the long-term memory technique at very low bit rates. The chapter concludes with a discussion in Section 6.4.

6.2 Multiple-Reference Motion Estimation: A Review

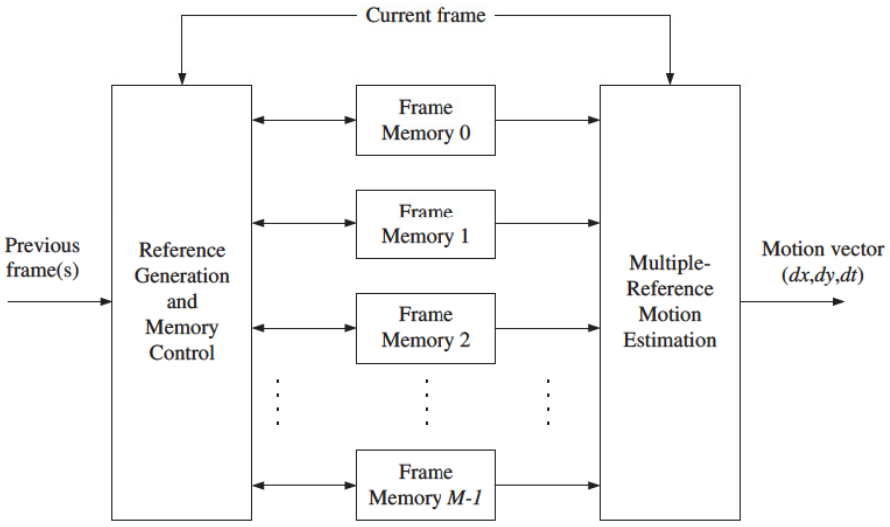
In *multiple-reference motion-compensated prediction* (MR-MCP), motion estimation and compensation are extended to utilize *more than one* reference frame. The reference frames are assembled in a *multi-frame memory* (or buffer) that is maintained simultaneously at encoder and decoder. In this case, in addition to the spatial displacements (d_x, d_y), a motion vector is extended to also include a *temporal displacement* d_t. This is the *index* into the multiframe memory. The process of MR-MCP is illustrated in Figure 6.1.

The main aim of MR-MCP is to improve coding efficiency. Thus, the reference generation block in Figure 6.1(a) can utilize any technique that provides useful data for motion-compensated prediction. Examples of such techniques are reviewed in what follows.

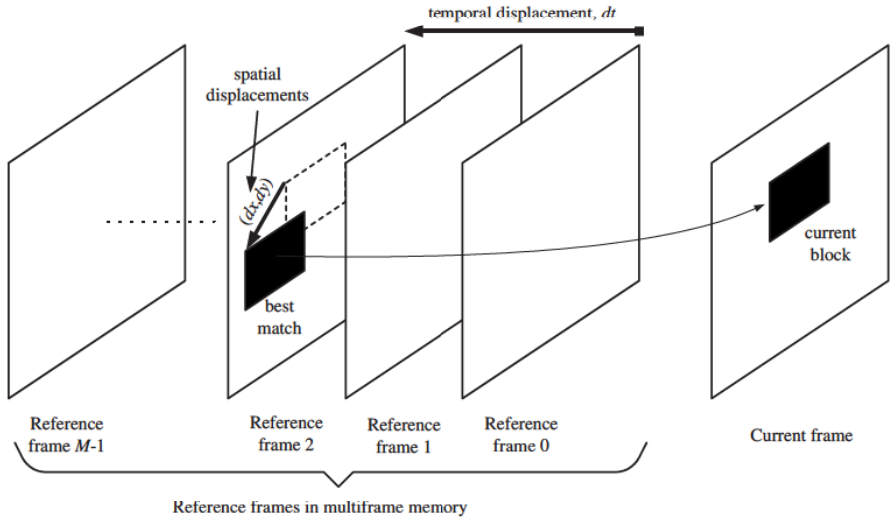
A number of MR-MCP techniques have been proposed for inclusion within MPEG-4. Examples are *global motion compensation* (GMC) [131, 132], *dynamic sprites* (DS) [132], and *short-term frame memory/long-term frame memory* (STFM/LTFM) prediction [133]. In these techniques, MCP is performed using two reference frames. The first reference frame is always the past decoded frame, whereas the second reference frame is generated using different methods. In GMC, the past decoded frame is warped to provide the second reference frame. The technique of DS is a more general case of GMC. In DS, past decoded frames are warped and blended into a sprite memory. This sprite memory is used to provide the second reference frame. In STFM/LTFM two frame memories are used. The STFM is used to store the past decoded frame, whereas the LTFM is used to store an earlier decoded frame. The LTFM is updated using a refresh rule based on scene-change detection. Both DS and STFM/LTFM can benefit from another MR-MCP technique, which is background memory prediction [134].

Similar to the STFM/LTFM is the reference picture selection (RPS) mode included in annex N of H.263+ (refer to Chapter 3). In this mode, switching to a different reference picture can be signaled at the picture level. It should be pointed out, however, that this option was designed for error resilience rather than for coding efficiency. Its main function is to stop error propagation due to transmission errors.

Probably the most significant contributions to the field of MR-MCP are those made by Wiegand and Girod *et al.* [135–141]. They noted [135, 136] that long-term statistical dependencies in video sequences are not exploited by existing video standards. Thus, they proposed to extend motion estimation and compensation to utilize several past decoded frames. They called this technique *long-term-memory motion-compensated prediction* (LTM-MCP). They demonstrated that the use of this technique can lead to significant improvements in coding efficiency.



(a) Multiple-reference motion estimation



(b) Multiple-reference motion compensation

Figure 6.1: Multiple-reference motion-compensated prediction

In Ref. 137 they proposed to use multiple global motion models to generate the reference frames. Thus, reference frames in this case are warped versions of the previously decoded frame using polynomial motion models. This can be seen as an extension to GMC, where, in addition to the most dominant global motion, less dominant motion is also captured by additional motion parameter sets. In order to determine the multiple models, a robust clustering method based on the iterative application of the least median of squares estimator is employed. This model estimation method is computationally expensive. In Ref. 138 they proposed an alternative method in which the past decoded frame is split into blocks of fixed size. Each block is then used to estimate one model using translational block matching followed by a gradient-based affine refinement. In addition to reduced complexity, this method leads to higher prediction gains.

In Ref. 139 they have demonstrated that combining the LTM-MCP method of Refs. 135 and 136 with the multiple GMC method of Ref. 138 can lead to further coding gains.

Recently, MR-MCP has been included in the enhanced reference picture selection (ERPS) mode (annex U) of H.263++ (refer to Chapter 3).

6.3 Long-Term Memory Motion-Compensated Prediction

As already discussed, there are many MR-MCP techniques. The main difference between those techniques is in the way they generate the reference frames. The simplest and least computationally complex approach is the LTM-MCP technique, where past decoded frames are assembled in the multiframe memory. This chapter will therefore concentrate on the LTM-MCP technique. More complex techniques, such as multiple GMC, may not be suitable for computationally constrained applications such as mobile video communication.

There are many ways to control the multiframe memory in the LTM-MCP technique. The simplest approach is to use a *sliding-window* control method. Assuming that there are M frame memories: $0 \ldots M \quad 1$, then the most recently decoded past frame is stored in frame memory 0, the frame that was decoded M time instants before is stored in frame memory $M - 1$, and so on. In the next time instant, the window is moved such that the oldest frame is dropped from memory, the contents of frame memories $0 \ldots M - 2$ are shifted to frame memories $1 \ldots M - 1$, and the new past decoded frame is stored in frame memory 0. According to this arrangement the new motion vector component is in the range $0 \leq d_t \leq M - 1$, where $d_t = 0$ refers to the most recent reference

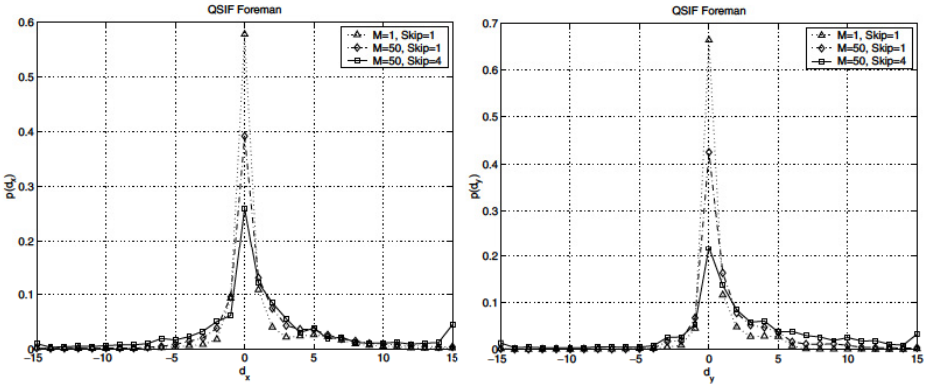
frame in memory. This sliding-window technique will be adopted throughout this chapter.

6.3.1 Properties of Long-Term Block-Motion Fields

This subsection investigates the properties of long-term block-motion fields and compares them to those of single-reference block-motion fields. All illustrations in this subsection were generated using a full-pel full-search long-term memory block-matching algorithm applied to the luma component of the FOREMAN sequence with blocks of 16×16 pels, a maximum allowed displacement of ± 15 pels, SAD as the distortion measure, restricted motion vectors, and original reference frames.

Property 6.3.1.1 *The distribution of the long-term memory spatial displacements* (d_x, d_y) *is center-biased.* This is evident from Figure 6.2, which shows the distribution of the relative frequency of occurrence of the spatial displacements d_x (Figure 6.2(a)) and d_y (Figure 6.2(b)). Note that this is similar to the single-reference case $(M = 1, \text{skip} = 1)$, although in the case of multiple-reference $(M = 50, \text{skip} = 1)$, the distribution is slightly more spread, which indicates that longer displacements are slightly more probable. This distribution is even more spread at higher frame skips, $(M = 50, \text{skip} = 4)$.

Property 6.3.1.2 *The distribution of the long-term memory temporal displacement* d_t *is zero-biased.* This is evident from Figure 6.3, where the temporal displacement $d_t = 0$ (which refers to the most recent reference frame



(a) Distribution of relative frequency of occurrence of d_x

(b) Distribution of relative frequency of occurrence of d_y

Figure 6.2: Center-biased distribution of the long-term memory spatial displacements (d_x, d_y)

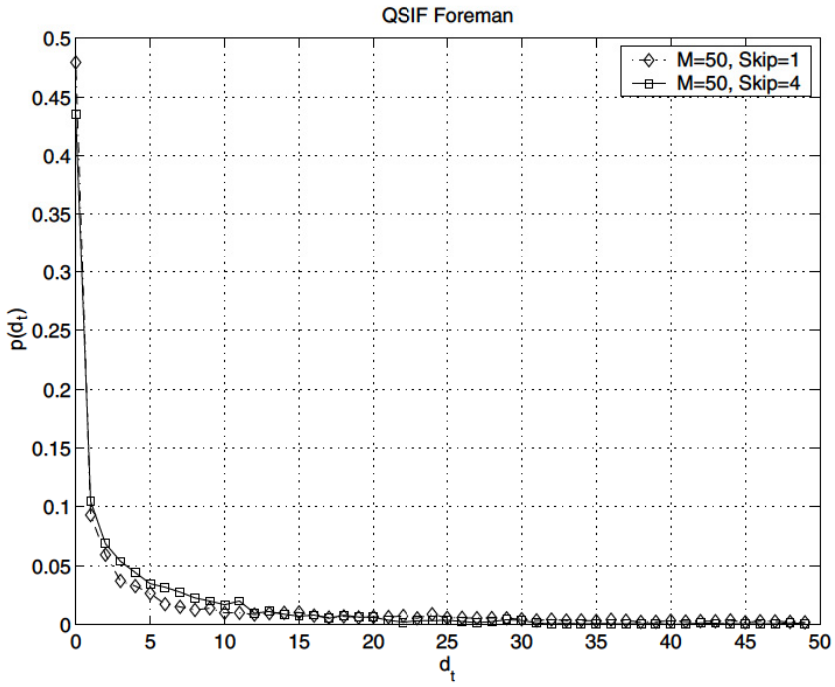
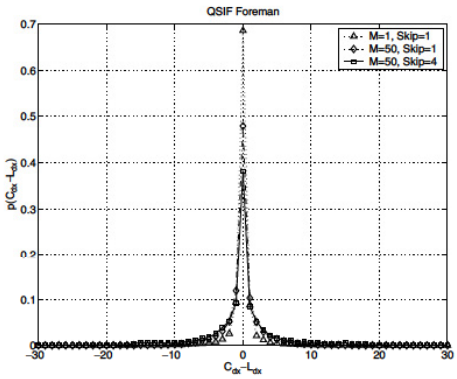


Figure 6.3: Zero-biased distribution of the long-term memory temporal displacement d_t

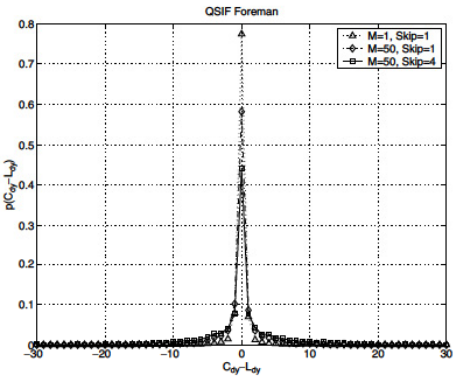
in memory) has the highest frequency of occurrence; and as the temporal displacement increases, its frequency of occurrence decreases. Note that this distribution becomes more spread at higher frame skips, which indicates that the selection of older reference frames becomes slightly more probable.

Property 6.3.1.3 *The long-term memory block-motion field is smooth and varies slowly*. In other words, there is high correlation between the motion vectors of adjacent blocks. This is evident from Figure 6.4, which shows the distribution of the difference between the current vector **C** and its left neighbor **L**. This is shown for the three components: d_x (Figure 6.4(a)), d_y (Figure 6.4(b)), and d_t (Figure 6.4(c)). All three distributions are biased toward a zero difference, which indicates high correlation. Note that this correlation is slightly less in the multiple-reference case ($M = 50, \text{skip} = 1$), compared to the single-reference case ($M = 1, \text{skip} = 1$). This correlation is further reduced at higher frame skips, ($M = 50, \text{skip} = 4$).

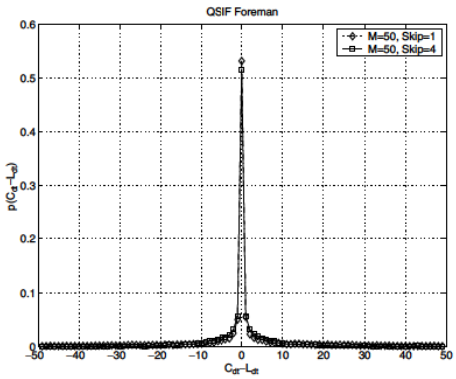
In general, it can be concluded that moving from a single-reference system to a multiple-reference system does not significantly change the properties of the block-motion field.



(a) Distribution of the difference between the horizontal component, d_x, of the current vector and its left neighbor



(b) Distribution of the difference between the vertical component, d_y, of the current vector and its left neighbor



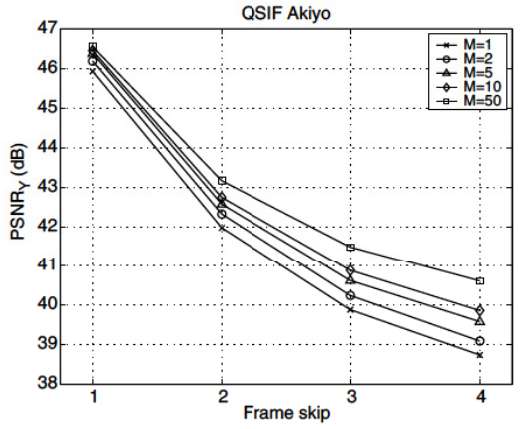
(c) Distribution of the difference between the temporal component, d_t, of the current vector and its left neighbor

Figure 6.4: Highly correlated long-term memory block-motion field

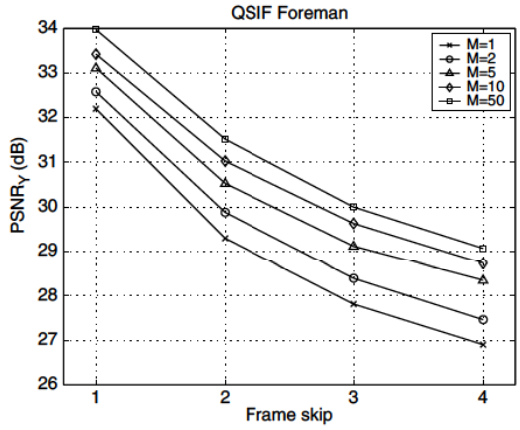
6.3.2 Prediction Gain

This subsection evaluates the prediction gain achieved by LTM-MCP. All results were generated using full-pel full-search long-term memory block matching with blocks of 16×16 pels, a maximum allowed displacement of ± 15 pels, SAD as the distortion measure, restricted motion vectors, and original reference frames. All quoted results refer to the luma components of sequences.

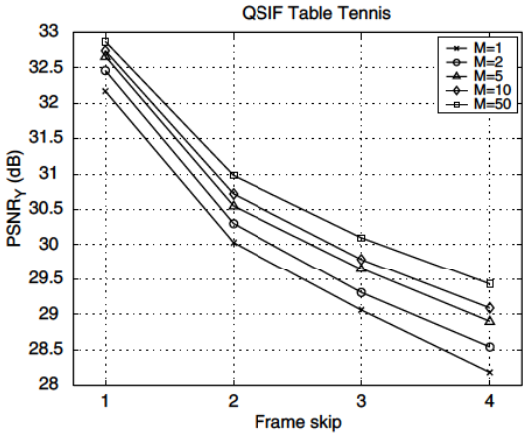
Figure 6.5 shows the performance of LTM-MCP when applied to the three QSIF sequences AKIYO, FOREMAN, and TABLE TENNIS with different memory sizes and different frame skips. It is immediately evident from this figure that significant prediction gains are achieved when utilizing more than one



(a) AKIYO



(b) FOREMAN



(c) TABLE TENNIS

Figure 6.5: Prediction quality of LTM-MCP with different memory sizes and frame skips

reference frame. For example, at a frame skip of 4, the prediction gain when using a multiframe memory of size $M = 50$ frames is 1.87 dB for AKIYO, 2.17 dB for FOREMAN, and 1.25 dB for TABLE TENNIS, compared to single-reference prediction (i.e., $M = 1$). Such prediction gains are mainly due to the long-term statistical dependencies of video sequences. Examples of such dependencies are the repetitions of sequence content due to uncovered objects or objects reappearing in the sequence. An interesting point to note here is that the prediction gains increase with increased frame skip. For example, for AKIYO when going from $M = 1$ to $M = 50$, the prediction gain is 0.62 dB at a frame skip of 1 and 1.87 dB at a frame skip of 4. This may be due to the fact that as the frame skip increases, successive frames get more decorrelated. This increases the chance that a frame other than the immediately preceding one will be chosen and, consequently, gives more chance to benefit from long-term memory prediction. In Ref. 136, the benefits of extending LTM-MCP to half-pel accuracy are discussed. It is shown that further prediction gains can be achieved by moving from full- to half-pel accuracy. This "accuracy gain" is comparable to that in the case of single-reference prediction.

It should be emphasized that the improved prediction quality of LTM-MCP is achieved at the expense of:

1. Increased memory requirements at both the encoder and the decoder.
2. Additional bit rate to transmit the new extra components, d_t, of motion vectors.
3. Increased computational complexity at the encoder.

Item 1 is not a major drawback due to the rapid drop in the price of memory chips, item 2 will be investigated further in Section 6.3.3, whereas a possible solution for item 3 will be proposed in Chapter 8.

6.3.3 Efficiency at Very Low Bit Rates

As already discussed in Section 6.1, LTM-MCP extends the motion vector of a block by a third component, d_t. This is the temporal displacement or the index into the multiframe memory. Obviously, the transmission of this extra component incurs an additional bit rate compared to the single-reference case. This additional bit rate has to be justified in terms of an improvement in the rate-distortion (R-D) performance. This subsection investigates the R-D performance of the LTM-MCP technique. Particular emphasis is given to the efficiency of this technique at the very low bit rates typical of mobile video communication. Four H.263-like encoders were implemented:

SR This is a single-reference encoder. It uses full-pel full-search block matching with macroblocks of 16×16 pels, a maximum allowed spatial

displacement of ± 15 pels, SAD as the distortion measure, restricted motion vectors, and reconstructed reference frames. Motion vectors are coded using the median predictor and the VLC table of the H.263 standard. The frame signal (in case of INTRA) and the DFD signal (in case of INTER) are transform encoded according to the H.263 standard. The encoder does not employ rate-constrained motion estimation and mode decision. Thus, motion estimation simply chooses the motion vector that minimizes the SAD measure without any bit-rate considerations. The INTRA/INTER decision is based on heuristic thresholds and is given by the following [142]:

$$\text{INTRA mode is chosen if} \qquad A < (\text{SAD}(\mathbf{d}) - 500), \tag{6.1}$$

where

$$A = \sum_{(x,y)\in\mathscr{B}} |f_c(x,y) - \bar{B}| \tag{6.2}$$

and

$$\bar{B} = \frac{\sum_{(x,y)\in\mathscr{B}} f_c(x,y)}{256}, \tag{6.3}$$

where $\mathbf{d} = (d_x, d_y)$ is the motion vector of macroblock $\mathscr{B}$ in the current frame f_c and SAD($\mathbf{d}$) is the SAD between the macroblock in the current frame and a corresponding macroblock in the reference frame shifted by $\mathbf{d}$.

SR-RC This is a single-reference rate-constrained encoder. It is the same as SR, but it uses rate-constrained motion estimation and mode decision as defined in the high-complexity mode of the H.263 test model, near-term, version 10 (TMN10) [142]. In this mode, motion estimation chooses the motion vector that minimizes the following Langrangian cost function:

$$J_{\text{MOTION}} = D_{\text{MOTION}} + \lambda_{\text{MOTION}} R_{\text{MOTION}}, \tag{6.4}$$

where D_{MOTION} is the SAD between the macroblock in the current frame and the corresponding macroblock in the reference frame shifted by $\mathbf{d}$, R_{MOTION} is the number of bits used to encode the motion vector $\mathbf{d}$, and λ_{MOTION} is a Lagrange multiplier related to the quantization parameter QP using

$$\lambda_{\text{MOTION}} = 0.92 \times \text{QP}. \tag{6.5}$$

To decide the mode, two Langrangian cost functions, one for each mode, are calculated as follows:

$$J_{\text{INTRA}} = D_{\text{INTRA}} + \lambda_{\text{MODE}} R_{\text{INTRA}}, \tag{6.6}$$

$$J_{\text{INTER}} = D_{\text{INTER}} + \lambda_{\text{MODE}} R_{\text{INTER}}, \tag{6.7}$$

where D_{INTRA} is the SSD between the current macroblock and its INTRA encoded reconstruction and R_{INTRA} is the number of bits used to INTRA encode the current macroblock. Similar definitions also apply for D_{INTER} and R_{INTER}, but they are calculated by INTER encoding the current macroblock. In both equations, λ_{MODE} is a Lagrange multiplier related to the quantization parameter QP using

$$\lambda_{\text{MODE}} - 0.85 \times \text{QP}^2. \tag{6.8}$$

The mode with the minimum cost function is chosen as the mode of the current macroblock. Note that, in this case, a macroblock needs to be encoded twice before being able to decide its mode. This increases the complexity of the encoder. A more detailed description of this rate-constrained motion estimation and mode decision method can be found in Ref. 143.

MR This is a multiple-reference encoder with no rate constraints. Thus, it is the same as SR, but it uses long-term memory motion-compensated prediction.

MR-RC This is a multiple-reference rate-constrained encoder. Thus, it is the same as SR-RC, but it uses long-term memory motion-compensated prediction.

The preceding encoders were tested using the three QSIF test sequences AKIYO, FOREMAN, and TABLE TENNIS. The frame skip parameter was set to 3 to achieve low bit rates. To generate a range of bit rates, the quantization parameter QP was varied over the range 5–30 in steps of 5. This means that each encoder was used to encode a given sequence six times. Each time, QP was held constant over the whole sequence (i.e., no rate control was used). The first frame was always INTRA encoded. The INTRA bits of the first frame were included in the bit-rate calculations, and no header bits were generated. All quoted results refer to the luma components of sequences. For MR and MR-RC, sliding-window control was used to maintain a long-term memory of size $M = 50$ frames. The VLC codewords in Table 6.1 were used to encode[1] the temporal components d_t of the long-term motion vectors.

Figures 6.6, 6.7, and 6.8 show the R-D performance of the preceding encoders for the three test sequences. Note that both single-reference and

[1]For example, since $d_t = 4$ is in the range (3:6), then according to Table 6.1 it will be encoded using a 5-bit codeword. This codeword is derived as follows. With reference to the start of its range, $d_t = 4$ is represented by $d_t - 3 = 4 - 3 = 1$. Thus, $x_1x_0 = 01$ and the codeword is given by $0x_1 1x_0 0 = 00110$.

Table 6.1: VLC codewords for encoding the temporal displacement d_t. Reproduced from Ref. 140

d_t	Bits	Codeword
0	1	1
"x_0" + 1 (1:2)	3	$0x_00$
"x_1x_0" + 3 (3:6)	5	$0x_11x_00$
"$x_2x_1x_0$" + 7 (7:14)	7	$0x_21x_11x_00$
"$x_3x_2x_1x_0$" + 15 (15:30)	9	$0x_31x_21x_11x_00$
"$x_4x_3x_2x_1x_0$" + 31 (31:62)	11	$0x_41x_31x_21x_11x_00$

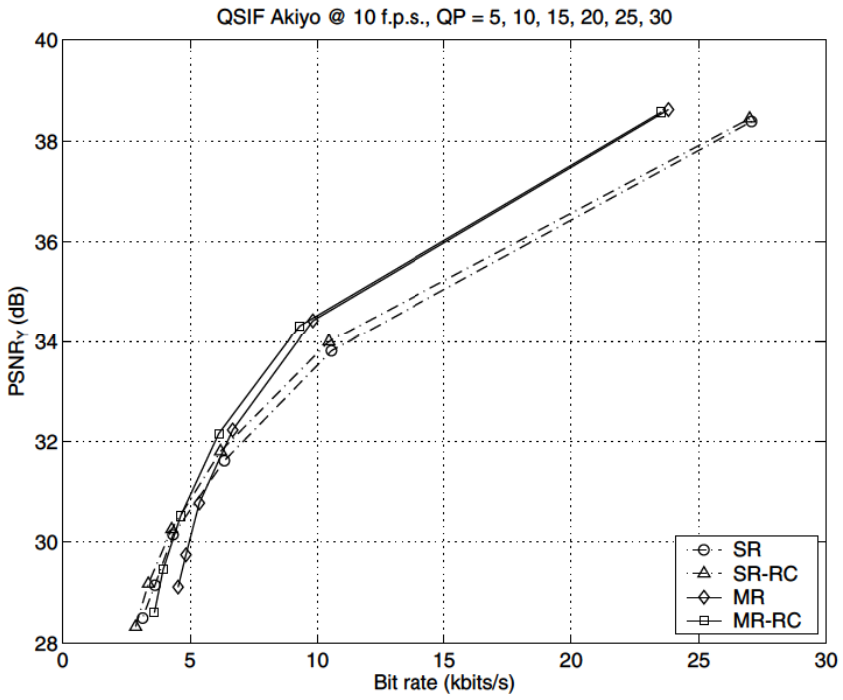


Figure 6.6: R-D performance of different single- and multiple-reference (with $M = 50$) encoders when encoding QSIF AKIYO at 10 frames/s

multiple-reference encoders benefit from the use of rate-constrained motion estimation and mode decision. Those benefits are more evident in high-movement sequences, where the use of more bits to encode the longer motion vectors has to be justified and controlled. It should be pointed out, however, that such benefits are achieved at the expense of increased computational complexity.

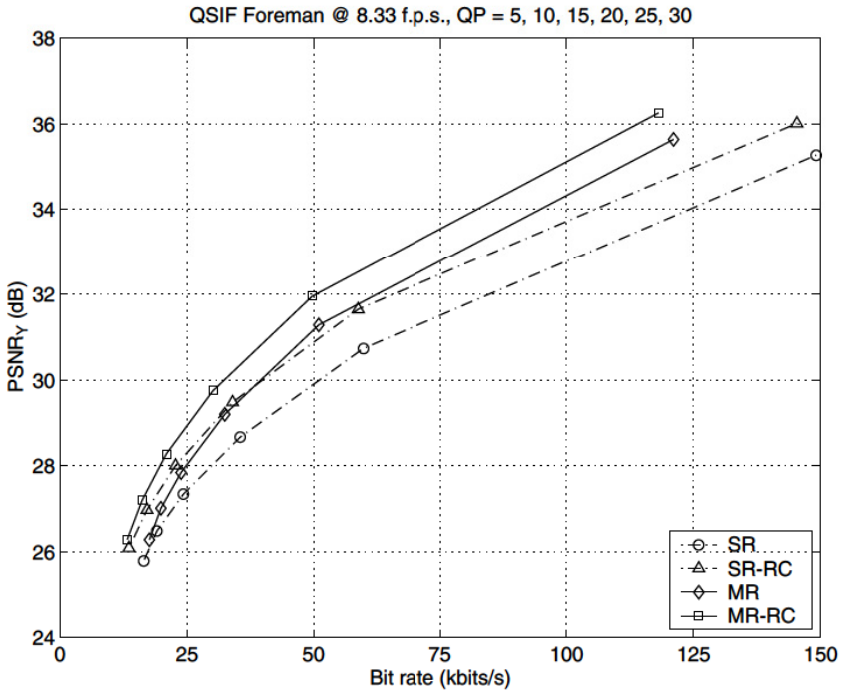


Figure 6.7: R-D performance of different single- and multiple-reference (with $M = 50$) encoders when encoding QSIF FOREMAN at 8.33 frames/s

Due to the additional bit rate generated by the temporal components d_t, the use of rate-constrained motion estimation and mode decision is essential in the case of multiple-reference encoders. A single-reference rate-constrained encoder (SR-RC) can outperform a multiple-reference encoder with no rate constraints (MR). This is evident at very low bit rates in Figures 6.6 and 6.7 and at all bit rates in Figure 6.8. In fact, at very low bit rates, even a single-reference encoder with no rate constraints (SR) can sometimes outperform the multiple-reference encoder (MR).

The best overall performance is achieved by the multiple-reference rate-constrained encoder (MR-RC). The benefits of this encoder become more evident as the bit rate increases. Note, however, that this improved performance is at the expense of a significant increase in computational complexity. This increase is due to the use of more than one reference frame during motion estimation and also to the use of rate-constrained motion estimation and mode decision. Note, also, that at extremely low bit rates a similar performance can be achieved by the less complex (SR-RC) encoder. Thus, at such bit rates the use of LTM-MCP is not justifiable.

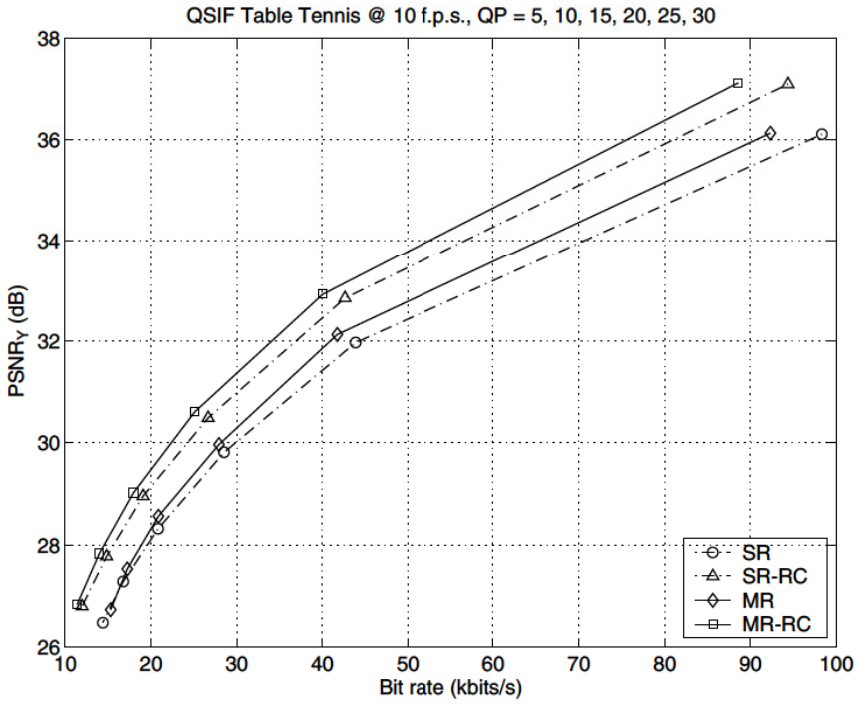


Figure 6.8: R-D performance of different single- and multiple-reference (with $M = 50$) encoders when encoding QSIF TABLE TENNIS at 10 frames/s

6.4 Discussion

Higher coding efficiency is one of the main requirements for mobile video communication. One way to achieve higher coding efficiency is to use advanced motion estimation techniques. One of the promising advanced techniques is multiple-reference motion-compensated prediction (MR-MCP).

This chapter reviewed the main efforts in the field of MR-MCP. It then investigated the performance of the long-term memory motion-compensated prediction (LTM-MCP) technique. It was found that this technique provides significant prediction gains compared to the single-reference case. It was realized, however, that such prediction gains are achieved at the expense of an additional bit rate to transmit one extra temporal component per motion vector. This additional bit rate has to be justified in terms of an improved rate-distortion (R-D) performance. An investigation into the R-D performance of LTM-MCP codecs revealed that the use of rate-constrained motion estimation and mode decision is important for the success of such techniques. Without rate constraints, the R-D performance of the LTM-MCP technique

can, at very low bit rates, drop below that of single-reference codecs. Combined with rate constraints, the LTM-MCP technique provides a superior R-D performance, which becomes more evident as the bit rate increases.

The chapter investigated the properties of long-term memory block-motion fields. It was found that the distribution of the long-term memory spatial displacements is center-biased. This distribution becomes more spread with increased frame memory size and frame skip. It was also found that the distribution of the long-term memory temporal displacement is zero-biased. Again, this distribution becomes more spread with increased frame memory size and frame skip. The investigation revealed also that the long-term memory block-motion field is highly correlated. In general, it was concluded that moving from a single-reference system to a multiple-reference system does not significantly change the properties of the block-motion field.

Part III
Computational Complexity

In mobile terminals, processing power and battery life are very limited and scarce resources. Given the significant amount of computational power required to process video, the use of reduced-complexity techniques is essential.

Motion estimation is the most computationally intensive process in a typical video codec. In fact, the computational complexity of this process is greater than that of all the remaining encoding steps combined. Thus, by reducing the complexity of this process, the overall complexity of the codec can be reduced.

This part contains two chapters. Chapter 7 reviews reduced-complexity motion estimation techniques. The chapter uses implementation examples and profiling results to highlight the need for reduced-complexity motion estimation. It then reviews some of the main reduced-complexity block-matching motion estimation techniques. The chapter then presents the results of a study comparing the different techniques.

Chapter 8 gives an example of the development of a novel reduced-complexity motion estimation technique. The technique is called the *simplex minimization search* (SMS). The development process is described in detail, and the SMS technique is then tested within an isolated test environment, a block-based H.263-like codec, and an object-based MPEG-4 codec. In an attempt to reduce the complexity of multiple-reference motion estimation (investigated in Chapter 6), the chapter then extends the SMS technique to the multiple-reference case. The chapter presents three different extensions (or algorithms) representing different degrees of compromise between prediction quality and computational complexity.

Chapter 7

Reduced-Complexity Motion Estimation Techniques

7.1 Overview

As already discussed, one of the main requirements for mobile video communication is *reduced-complexity*. It is not difficult to show that the high computational complexity of a typical video codec is due mainly to the *motion estimation* process. Thus, by reducing the complexity of this process, the overall complexity of the codec can be reduced. This chapter reviews reduced-complexity motion estimation techniques. In particular, the chapter concentrates on reduced-complexity block-matching motion estimation (BMME) techniques. The chapter also presents the results of a study comparing different reduced-complexity BMME techniques.

The rest of the chapter is organized as follows. Section 7.2 uses implementation examples and profiling results to highlight the need for reduced-complexity motion estimation. Sections 7.3–7.7 review the main categories of reduced-complexity BMME algorithms. Section 7.8 presents the results of a study comparing the different categories. The chapter concludes with a discussion in Section 7.9.

7.2 The Need for Reduced-Complexity Motion Estimation

Processing digital video requires a significant amount of computational power. This represents one of the main challenges for real-time mobile video communication, where processing power and battery life are scarce resources. For example, an MPEG-4 simple profile codec has recently been implemented

on Texas Instruments' TMS320C541 40-MHz processor [5].[1] Profiling results show that this codec cannot achieve real-time processing even when using SQCIF sequences. It can encode only about 1 frame/s, and it can decode only about 20 frames/s. Another example is the implementation of the H.263 baseline mode on the more powerful TMS320C62 200-MHz processor, as described in Ref. 6. Again, this implementation cannot achieve real-time processing, for it only can encode about 5 QCIF frames/s.

Looking at the building blocks of a typical video codec, it is not difficult to realize that this huge computational complexity is due mainly to the motion estimation process. As already discussed, most video codecs estimate motion using the *block-matching motion estimation* (BMME) algorithm. The most straightforward BMME algorithm is the *full-search* (FS) algorithm, sometimes referred to as the *exhaustive search* or the *brute-force search*. This algorithm is guaranteed to find the best-match block because it exhaustively searches over all possible blocks (search locations or candidate motion vectors) within the search window. The algorithm produces the best possible prediction quality. This is, however, at the expense of a huge computational complexity. For example, for a CIF video sequence encoded at 30 frames/s with 16×16 blocks, maximum displacement of ± 15 pels, and SAD as the distortion measure, a direct implementation of a full-pel FS-BMME algorithm requires about 6×10^9 integer additions and subtractions, 3×10^9 magnitude operations, and 11×10^6 comparisons per second. In fact, the computational complexity of this motion estimation process is greater than that of all the remaining encoding steps combined. This is clear from Table 7.1, which shows profiling results[2] of the baseline mode of Telenor's H.263 encoder [144] when used to encode the QCIF FOREMAN sequence at 64 kbits/s. In this case, the motion estimation process[3] consumes about 70% of the overall encoding time.

Because of this high computational complexity, motion estimation has become a bottleneck problem in many applications, e.g., mobile video terminals and software-based video codecs, especially if real-time video coding is required. This has motivated the development of a number of fast motion estimation algorithms since the early 1980s. In fact, recent advances in video coding not only highlight the importance of such algorithms, but even call for further research into the area of reduced-complexity motion estimation. For example, HDTV and multiple-reference motion estimation (discussed

[1] According to Ref. 5, about half of all mobile phones currently use a 'C54x processor.

[2] The results were obtained using the profiler of the Visual C++ 5.0 compiler run on a PC with a Pentium 100-MHz processor, 64 MB of RAM, and a Windows 98 operating system.

[3] The baseline mode of Telenor's H.263 codec uses block matching with 16×16 blocks, SAD as the distortion measure, and ± 15 pels maximum displacement. Full-pel accuracy is first obtained using full search. This is then refined to half-pel accuracy.

Table 7.1: Profiling results of Telenor's H.263 baseline mode when used to encode QCIF FOREMAN at 64 kbits/s

Function	CPU Time (ms)	Percentage (%)
Motion estimation	240,354	69.7
Input/output	32,552	9.4
DCT/IDCT	29,412	8.5
Others	42,353	12.4
Total	344,671	100.0

in Chapter 6) have a computational complexity that is several orders of magnitude higher than that shown in the preceding examples. The former uses higher spatial resolutions and larger search windows, and the latter extends the search over several reference frames.

The following sections of this chapter review the main categories of reduced-complexity BMME algorithms. Although each category can be used on its own, careful encoder design can utilize different combinations to achieve higher speed-up ratios.

7.3 Techniques Based on a Reduced Set of Motion Vector Candidates

Instead of searching over all possible blocks within the search window, this category restricts the search over a selected subset of the blocks. Most algorithms in this category are, implicitly or explicitly, based on the *unimodal error surface assumption* [54], which states that *the block distortion measure (BDM) increases monotonically as the search location moves away from the best-match location*. Therefore, the search starts by evaluating the BDM at locations coarsely spread over the search window according to some predefined uniform pattern. This is then repeated with finer resolution (i.e., smaller spread) around the search location with the minimum BDM from the preceding step.

The first algorithm reported in this category was the *two-dimensional logarithmic* (TDL) search proposed in 1981 by Jain and Jain [54]. Figure 7.1 shows an example of the TDL search with a maximum displacement of $d_m = 6$ pels. The search is initialized at the origin of the search window with a suitable step size (i.e., the spacing between the search locations). In each step of the TDL search, the BDM is evaluated at five search locations. In the given example, the search locations $(0,0)$, $(+2,0)$, $(-2,0)$, $(0,+2)$, $(0,-2)$ form

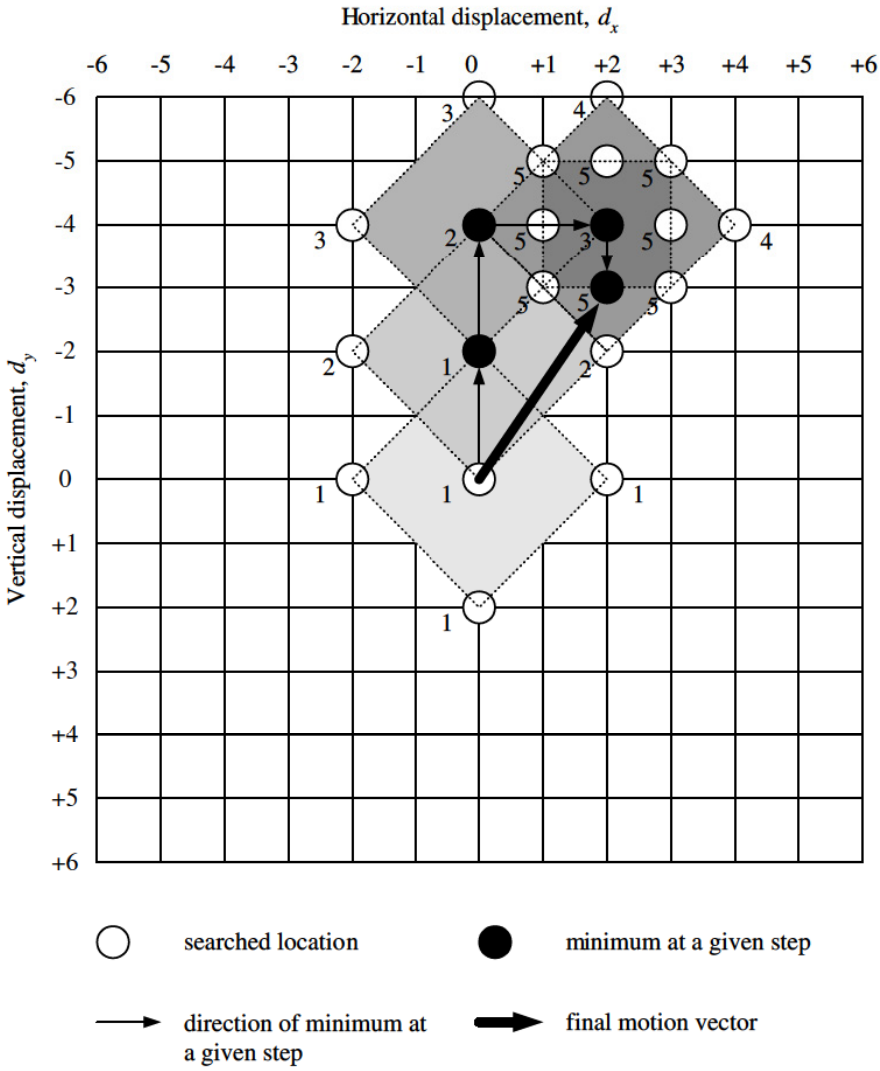


Figure 7.1: An example of the TDL search with $d_m = 6$ pels

the search pattern of the first step. At each step, the search pattern is centered around the minimum of the previous step. In the given example, the minimum in the first step is at $(0, -2)$. Thus, the search pattern in the second step is centered around this minimum location. The step size is reduced by a factor of 2 if the minimum is in the center of the search pattern or at the boundary of the search window. In the fourth step of the given example, the minimum is at $(+2, -4)$, which is the center of the search pattern. Therefore, the spacing between the search locations is halved in the fifth step. Since halving

the distance between the search locations gives a step size of 1, this indicates that this is the final step in the search. In this final step, BDM is evaluated at the minimum from the previous step and also at its eight nearest neighbors. In the given example, the final motion vector is $(+2, -3)$.

Since the introduction of the TDL search, a large number of similar algorithms have been proposed. Examples are the *three-step search* (TSS) [145], the *one-at-a-time search* (OTS) [146], the *conjugate-directions search* (CDS) [146], the *cross-search algorithm* (CSA) [147], the *genetic motion search* (GMS) [148], and the *diamond search* (DS) [149–151], to mention a few. The appendix gives a detailed description of some of these algorithms.

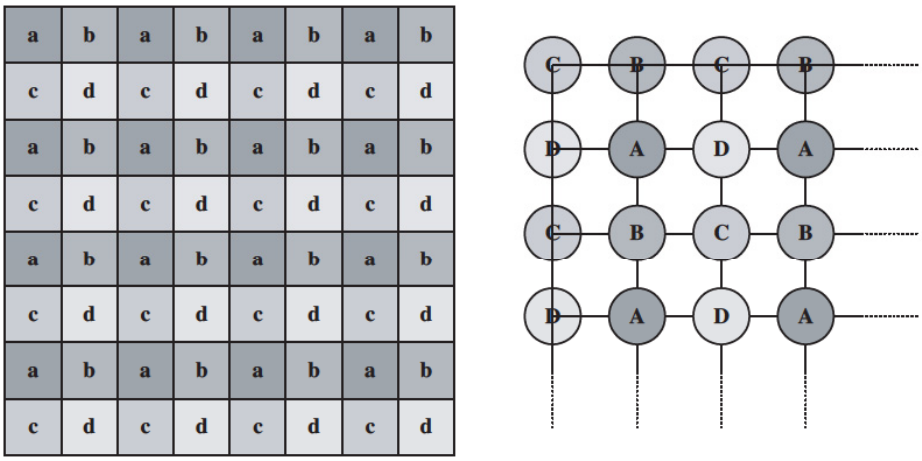
Compared to other techniques, this category of techniques provides a relatively high speed-up ratio and has, therefore, received most of the attention. However, as is shown later, the unimodal error surface assumption does not always hold true, and such algorithms can easily be trapped in local minima, resulting in a suboptimal prediction quality.

7.4 Techniques Based on a Reduced-Complexity Block Distortion Measure

In this category, reduced-complexity is achieved by employing a reduced-complexity BDM. As already discussed in Chapter 4 (Section 4.6.1), most implementations prefer the SAD measure, due to its reduced-complexity and good prediction quality. A number of other reduced-complexity BDMs have also been proposed in the literature. Examples are the *pel difference classification* (PDC) [152], the *minimized maximum* (MiniMax) error [153], the *reduced-bits mean absolute difference* (RBMAD) [154], *integral projections* [155], and *one-bit/pixel* [156], to mention a few. Most of these measures were designed specifically for efficient hardware and VLSI implementation, but their prediction quality is not as good as the SSD or the SAD measures.

Another type of algorithms in this category reduces the complexity of the BDM by subsampling the matched blocks. Obviously, since only a fraction of the pels is used in the matching process, this category does not guarantee to find the best match, even when combined with full search. Koga *et al.* [145] subsample the matched blocks by a factor of 2 both horizontally and vertically (i.e. 4:1 subsampling), reducing the complexity of the BDM by a factor of 4.

Instead of using a uniform subsampling pattern, Liu and Zaccarin [157] use alternating subsampling patterns. The patterns are alternated over the searched locations in such a way that effectively all pels of a block contribute to the matching process. This method is illustrated in Figure 7.2. Figure 7.2(a) shows an 8×8 block of pels. Four 4:1 subsampling patterns are defined in this block. For example, subsampling pattern **A** consists of all pels labeled **a** in the block.



(a) Four 4:1 subsampling patterns

(b) Alternating schedule of four subsampling patterns over the search window

Figure 7.2: Reduced-complexity BDM using the alternating subsampling patterns of Liu and Zaccarin [157]

Similarly, patterns **B**, **C**, and **D** consist of all the **b**, **c**, and **d** pels, respectively. Figure 7.2(b) shows part of the search window in the reference frame. Each circle in this figure represents a search location (i.e., a candidate block) in the window. During motion estimation, search locations labeled **A** use the subsampling pattern **A**, and so on. For each of the four subsampling patterns, the motion vector with the minimum BDM over the locations where that pattern is used is selected. For each of the four selected motion vectors, the BDM is evaluated, but this time without subsampling. The vector that achieves the minimum BDM is selected as the motion vector of the block. Compared to the approach of Koga *et al.*, this approach achieves approximately the same reduction in complexity, but with better prediction quality.

Chan and Siu [158] vary the number of pels in the subsampling pattern according to block details. Thus, fewer pels are used for uniform blocks and more pels are used for high-activity blocks. In this algorithm, the reduction in complexity varies between blocks and the prediction quality is generally better than that of Liu and Zaccarin.

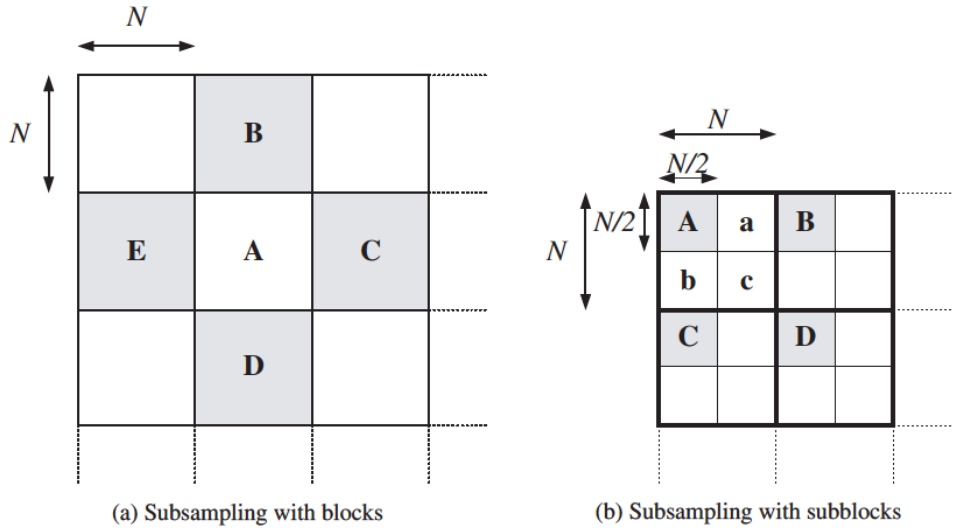
7.5 Techniques Based on a Subsampled Block-Motion Field

This category is based on the fact that block-motion fields of typical video sequences are usually smooth and vary slowly (as was shown in Section 4.6.7).

In other words, it is very common to find neighboring blocks with identical or nearly identical motion vectors. Thus, in this category, a subsampled block-motion field is first obtained by estimating the motion vectors for a fraction of the blocks in the frame. This field is then appropriately interpolated to determine the motion vectors of the remaining blocks.

Liu and Zaccarin [157] use a checkerboard subsampling pattern and estimate the motion vectors for half of the blocks (i.e., a 2:1 subsampled motion field) using full search. Then they estimate the motion vectors of the other half using a limited search that examines only four candidate motion vectors. Those candidates are the four surrounding motion vectors that were estimated using full search. For example, in Figure 7.3(a) the motion vectors of blocks **B**, **C**, **D**, and **E** are estimated using full search. Only those four vectors are then used as candidates when estimating the motion of block **A**. This algorithm reduces complexity by roughly a factor of 2, with only a slight loss in prediction quality.

Another algorithm was also proposed by Liu and Zaccarin in Ref. 157. In this algorithm, each block is divided into four subblocks. Motion is first estimated, using full search, for one subblock in each block, say, the top-left subblock. The motion vectors of the remaining subblocks are then estimated using a limited search with candidates from the neighboring full-search motion vectors. For example, in Figure 7.3(b) the motion vectors of subblocks **A**, **B**, **C**, and **D** are estimated using full search. Only those four vectors are then used as candidates when estimating the motion vectors of subblocks **a**, **b**,



(a) Subsampling with blocks (b) Subsampling with subblocks

Figure 7.3: Reduced-complexity using the subsampled motion fields of Liu and Zaccarin [157]

and **c**. This algorithm reduces complexity by roughly a factor of 4. Since smaller blocks are employed, the algorithm provides better prediction quality than full-search with original size blocks. This is, however, at the expense of a larger motion overhead.

7.6 Hierarchical Search Techniques

This category uses a multiresolution representation of video. The basic idea is to perform motion estimation at each level successively, starting with the lowest resolution level. Thus, motion estimation is first performed at the lowest resolution level to obtain a rough estimate of the motion vector. This vector is then passed to the next-higher resolution level to serve as an initial estimate. Motion estimation at the higher resolution level is then used to refine this initial estimate. This process is repeated until the highest resolution level is reached. At lower resolution levels, smaller blocks are used for block matching. This reduces the complexity of calculating BDMs. At higher-resolution levels, smaller search ranges are used since motion estimation starts from a good initial estimate. This reduces the number of locations to be searched. Both factors (i.e., smaller blocks at low resolutions and smaller search ranges at high resolutions) contribute to reducing the overall complexity of the search. Note that when reducing the resolution of the searched frames, the motion speed is also reduced. This makes hierarchical techniques particularly useful for estimating, with reduced complexity, high motion content. Examples of hierarchical motion estimation algorithms are reported in Refs. 159 and 160.

Figure 7.4 shows an example of a three-level hierarchical motion estimation technique applied to a QCIF sequence. In this case, the current frame is first used to generate three current frames with the resolutions: 44×36, 88×72, and 176×144. Each resolution level is a low-pass filtered and subsampled version of the next-higher resolution level. The resulting representation is called a *mean pyramid*. The same process is also applied to the reference frame (i.e., the previous frame). Motion estimation starts at the lowest resolution level with a block size of 4×4 pels and a search range of ± 3 pels. The estimated motion vector of a block in this resolution is scaled up by a factor of 2 (i.e., the scaled vector will have a maximum range of $2 \times (\pm 3) = \pm 6$ pels) and then be passed to the corresponding block in the next-higher resolution level. Motion estimation in the next-higher resolution level uses a block size of 8×8 pels and a search range of ± 1 pel around the propagated vector from the lower resolution level. This produces a motion vector with a maximum range of $(\pm 6) + (\pm 1) = \pm 7$ pels, which is again scaled up by a factor of 2 (to a maximum range of ± 14) and propagated to the next-higher resolution level. In this level, a block size of 16×16 pels is used with a search range

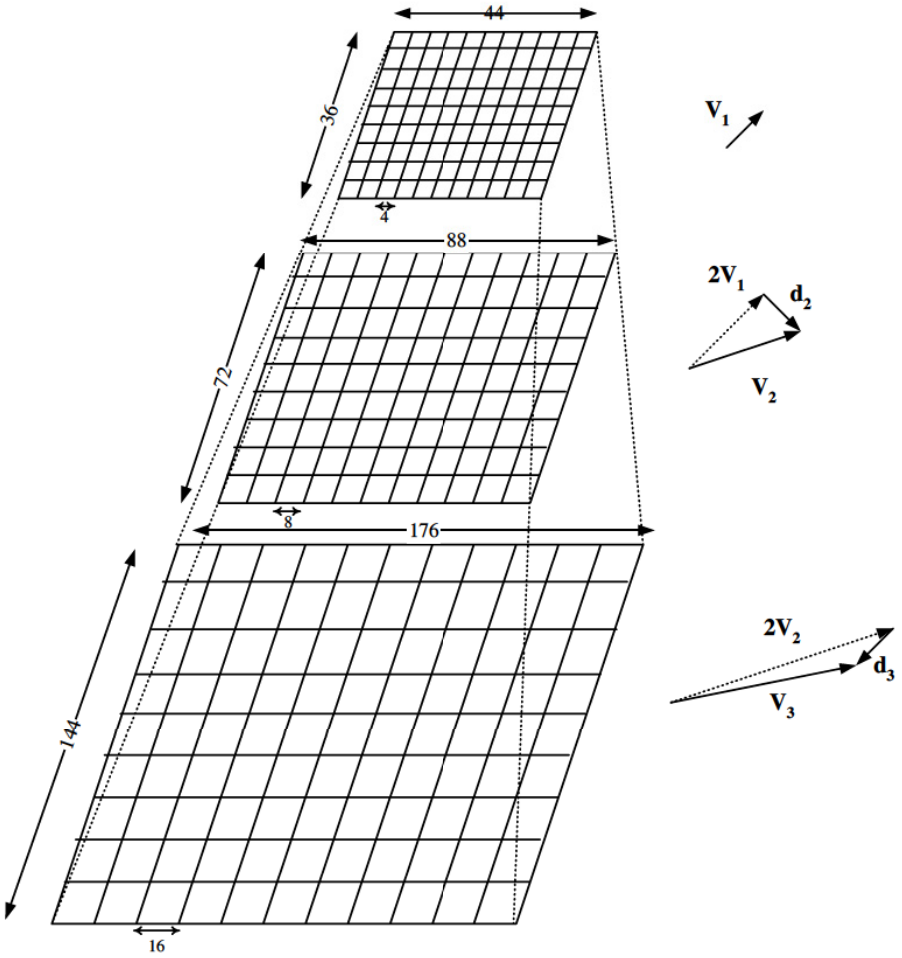


Figure 7.4: Hierarchical motion estimation using a mean pyramid of three levels applied to a QCIF frame

of ± 1 pel around the propagated vector from the lower resolution level. This gives a final vector with a maximum range of ± 15 pels.

There are many variants to hierarchical motion estimation. Some techniques use the same frame size in all levels of the hierarchy, with larger block sizes at lower levels. Other techniques use the same block size in all levels of the hierarchy, with subsampled frames at lower levels. In both cases, any level will have fewer blocks than the next higher level. Thus, a motion vector estimated at one level will be propagated to more than one block in the higher level.

In addition to reduced-complexity and robust estimation of high-motion content, hierarchical motion estimation algorithms are also reported to provide

more homogeneous block-motion fields and a better representation of the true motion in the frame [159]. The latter property is particularly important for motion-compensated interpolation.

7.7 Fast Full-Search Techniques

All the preceding categories reduce the computational complexity of the BMME process at the expense of a suboptimal prediction quality. This category, however, reduces complexity without sacrificing prediction quality. It is interesting to note that algorithms in this category are usually based on ideas borrowed from the field of fast codebook search for vector quantization (VQ).

An example of the algorithms in this category is the *partial distortion elimination* (PDE) algorithm. Assume that during a full search, the minimum BDM calculated so far is BDM(i_m, j_m) at search location (i_m, j_m). Then the BDM calculation of any subsequent search location (i, j) is stopped as soon as the accumulated distortion exceeds BDM(i_m, j_m). This idea is very similar to the fast-search VQ method reported in Ref. 161. Clearly, initializing the search at a location with the lowest possible BDM(i_m, j_m), achieves the highest possible reduction in computational complexity. As already shown in Section 4.6.7, the distribution of the best-match location is usually center-biased (i.e., the vector $(0, 0)$ has the highest probability, and longer vectors are less probable). Thus, the PDE algorithm is usually combined with a spiral-ordered search starting at the origin of the search space and going outward in a spiral fashion. This combination is employed, for example, in Telenor's H.263 codec [144].

Another algorithm in this category is the *successive elimination algorithm* (SEA) [162]. Again, this algorithm has similarities with the fast-search VQ algorithm reported in Ref. 163. The SEA algorithm is based on the triangular mathematical inequality given by

$$\left| \sum_{i=1}^{k} a_i \right| \leq \sum_{i=1}^{k} |a_i|, \tag{7.1}$$

where a_i are arbitrary real numbers. Extending this inequality to the SAD equation gives

$$\left| \sum_{(x,y)\in\mathcal{B}} |f_t(x, y)| - \sum_{(x,y)\in\mathcal{B}} |f_{t-\Delta t}(x - i, y - j)| \right|$$

$$\leq \sum_{(x,y)\in\mathcal{B}} |f_t(x, y) - f_{t-\Delta t}(x - i, y - j)|. \tag{7.2}$$

The first summation in this inequality is the sum norm of block $\mathscr{B}$ in the current frame, and this sum is denoted SN_t. The second summation, on the other hand, is the sum norm of a candidate block in the reference frame shifted by (i,j), and this sum is denoted $\mathrm{SN}_{t-\Delta t}(i,j)$. The third summation is obviously the $\mathrm{SAD}(i,j)$. Thus, for simplicity, Inequality (7.2) can be rewritten as

$$|\mathrm{SN}_t - \mathrm{SN}_{t-\Delta t}(i,j)| \leq \mathrm{SAD}(i,j). \tag{7.3}$$

Now assume that during a full-search, the minimum SAD calculated so far is $\mathrm{SAD}(i_m,j_m)$ at search location (i_m,j_m). A subsequent search location (i,j) is said to achieve better match only if $\mathrm{SAD}(i,j) \leq \mathrm{SAD}(i_m,j_m)$. Put in another way, and based on Inequality (7.3), a subsequent search location (i,j) is said to achieve better match only if $|\mathrm{SN}_t - \mathrm{SN}_{t-\Delta t}(i,j)| \leq \mathrm{SAD}(i_m,j_m)$. In other words, a subsequent location (i,j) can be immediately skipped from the search if

$$|\mathrm{SN}_t - \mathrm{SN}_{t-\Delta t}(i,j)| \geq \mathrm{SAD}(i_m,j_m). \tag{7.4}$$

Note that calculating the sum norms in this inequality has a reduced-complexity compared to calculating the $\mathrm{SAD}(i,j)$ itself. For example, assume that $\mathscr{B}(x,y)$ is an $N \times N$ block with its top-left cornet at (x,y) and that the next block $\mathscr{B}(x+1,y)$ is the block obtained by moving one pel to the right. The two blocks are overlapping and they share $N-1$ columns. Once the sum norm, $\mathrm{SN}(\mathscr{B}(x,y))$, of the first block is calculated, the sum norm, $\mathrm{SN}(\mathscr{B}(x+1,y))$, of the next block in the line is obtained simply by substracting the sum norm of the first column of block $\mathscr{B}(x,y)$ and adding the sum norm of the last column in block $\mathscr{B}(x+1,y)$. A similar procedure can be used for calculating the sum norm of the next block in the column (i.e., when moving one pel down). Based on these ideas, a very fast method of calculating the sum norms is presented in Ref. 162.

A similar algorithm to the SEA has also been proposed in Ref. 164. Assume that $\mathscr{B}$ is partitioned into subsets $\mathscr{B}_n$ such that $\mathscr{B} = \bigcup_n \mathscr{B}_n$ and $\bigcap_n \mathscr{B}_n = \emptyset$. Then the triangular inequality becomes

$$\sum_n |\mathrm{SN}_{t,n} - \mathrm{SN}_{t-\Delta t,n}(i,j)| \leq \mathrm{SAD}(i,j), \tag{7.5}$$

where $\mathrm{SN}_{t-\Delta t,n}(i,j)$ is the sum norm over subset $\mathscr{B}_n$ of the candidate block in the reference frame shifted by (i,j). It can be shown that

$$\sum_n |\mathrm{SN}_{t,n} - \mathrm{SN}_{t-\Delta t,n}(i,j)| \geq |\mathrm{SN}_t - \mathrm{SN}_{t-\Delta t}(i,j)|. \tag{7.6}$$

Thus, a tighter bound is achieved when the partitioned case is used in Inequality (7.4) instead of the partitioned case. This tighter bound will result

in faster rejection of more candidates and consequently will achieve higher speed-up ratios. However, the partitions must be chosen carefully to minimize the overhead calculations. In Ref. 164 two partitions are proposed. For an $N \times N$ block, the first partition produces N subsets, each being one of the N rows of the block, whereas the second partition produces N subsets, each being one of the N columns of the block. Again, this algorithm has similarities with the fast-search VQ algorithm presented in Ref. 47.

7.8 A Comparative Study

This section presents the results of a study comparing the categories of reduced-complexity motion estimation techniques discussed in Sections 7.3–7.7. The main aim of this study is to provide the reader with a feel of the relative performance of the discussed categories. Particular attention is given to the tradeoff between computational complexity and prediction quality.

In this study, one representative of each category was chosen. All simulated algorithms use 16×16 blocks, SAD as the distortion measure, ± 15 maximum displacement, full-pel accuracy, restricted motion vectors, and original reference frames. The simulated algorithms are:

FSA This is a full-search algorithm.

TDL This is the two-dimensional logarithmic search of Jain and Jain [54]. The algorithm is discussed in Section 7.3 and described in detail in the appendix, Section A.2.

SDM This algorithm uses a 4:1 subsampling of the matched blocks to reduce the complexity of calculating the distortion measure. The subsampling pattern used corresponds to pattern **A** described in Section 7.4. This pattern consists of all pels labeled **a** in Figure 7.2(a).

SMF This is the subsampled motion field algorithm of Liu and Zaccarin [157]. The algorithm is discussed in Section 7.5.

HME This is a three-level hierarchical motion estimation algorithm. The algorithm is described in Section 7.6 and illustrated using Figure 7.4.

PDE This is the partial distortion elimination algorithm described in Section 7.7. In order to reduce the overhead of logical operations, the condition to reject a given candidate is tested after accumulating the BDM of each row of the block (rather than after each pel of the block). The algorithm is supported with a spiral-ordered search starting at $(0, 0)$ and going outward toward longer motion vectors.

Tables 7.2–7.4 present the results of testing these algorithms using the three test sequences AKIYO, FOREMAN, and TABLE TENNIS, with a frame skip of 1 (i.e., 30 frames/s for AKIYO and TABLE TENNIS and 25 frames/s for FOREMAN). All results are averages over sequences and refer to the luma components. Each table compares the algorithms in terms of prediction quality and computational complexity. The prediction quality is presented in terms of average luma PSNR in decibels. The difference in PSNR between each algorithm and the FSA is also shown.[4] The computational complexity is presented in terms of the average motion estimation time (in milliseconds) per frame.[5] Care

Table 7.2: Comparison between different fast block-matching algorithms when applied to QSIF AKIYO at 30 frames/s

	Prediction quality		Computational complexity	
	PSNR (dB)	ΔPSNR (dB)	ME Time (ms/frame)	Speed-up ratio
FSA	45.93	0.00	1013.87	1.00
PDE	45.93	0.00	48.49	20.91
SDM	45.93	0.00	278.25	3.64
SMF	45.93	0.00	511.51	1.98
TDL	45.93	0.00	26.82	37.80
HME	45.93	0.00	20.73	48.89

Table 7.3: Comparison between different fast block-matching algorithms when applied to QSIF FOREMAN at 25 frames/s

	Prediction quality		Computational complexity	
	PSNR (dB)	ΔPSNR (dB)	ME Time (ms/frame)	Speed-up ratio
FSA	32.20	0.00	1258.95	1.00
PDE	32.20	0.00	149.80	8.40
SDM	31.96	−0.24	346.72	3.63
SMF	31.91	−0.29	634.08	1.99
TDL	31.80	−0.40	34.76	36.22
HME	31.88	−0.32	25.73	48.92

[4] ΔPSNR = PSNR of fast algorithm − PSNR of FSA.

[5] Motion estimation times were obtained using the profiler of the Visual C++ 6.0 compiler run on a PC with a Pentium-III 700-MHz processor, 128 MB of RAM, and a Windows 98 operating system.

Table 7.4: Comparison between different fast block-matching algorithms when applied to QSIF TABLE TENNIS at 30 frames/s

	Prediction quality		Computational complexity	
	PSNR (dB)	ΔPSNR (dB)	ME time (ms/frame)	Speed-up ratio
FSA	32.17	0.00	1049.11	1.00
PDE	32.17	0.00	125.02	8.39
SDM	31.99	−0.18	287.73	3.65
SMF	31.44	−0.73	529.00	1.98
TDL	31.63	−0.54	28.66	36.61
HME	31.85	−0.32	21.62	48.54

should be taken when interpreting the results because the motion estimation time can vary with implementation and the underlying hardware platform. The speed-up ratio of each algorithm with reference to the FSA is also shown.[6]

As expected the FSA provides the best prediction quality, but at the expense of a high computational complexity.

The PDE algorithm provides an identical prediction quality to FSA, with a moderate speed-up ratio. Note that the computational complexity of PDE is highly dependent on the type of sequence and the motion content. For example, most blocks in the AKIYO sequence are stationary or quasi-stationary. Since PDE is initialized at $(0,0)$, this will lead to a very low starting minimum value BDM(i_m, j_m). This will result in faster rejection of more candidates and, consequently, will lead to a relatively high speed-up ratio.

The SDM provides the next-best prediction quality. However, its 4:1 subsampling pattern limits its speed-up ratio to about 4. Similarly, the 2:1 field subsampling pattern of SMF limits its speed-up ratio to about 2. Note that the prediction quality of SMF is dependent on the amount of correlation between the motion vectors of neighboring blocks. This may explain the relatively high loss of prediction quality for the TABLE TENNIS sequence.

The TDL and HME algorithms provide the highest speed-up ratios, with moderate losses in prediction quality. In general, however, the HME algorithm outperforms the TDL algorithm in terms of both prediction quality and computational complexity.

[6] $\text{Speed-up} = \dfrac{\text{ME time for FSA}}{\text{ME time for fast algorithm}}$.

7.9 Discussion

Processing digital video requires a significant amount of computational power. This represents one of the main challenges for real-time mobile video communication, where processing power and battery life are scarce resources.

In this chapter, the computational complexity of a typical video codec was investigated. It was found that this complexity is due mainly to the motion estimation process. In fact, it was found that the computational complexity of this process is greater than that of all the remaining encoding steps combined. It was concluded, therefore, that reducing the complexity of this process is the best way to reduce the overall complexity of the codec. The chapter reviewed the main categories of reduced-complexity BMME techniques. The chapter then presented the results of a study comparing the different categories. It was found that hierarchical techniques and techniques based on a reduced set of motion vector candidates, in general, provide the highest reduction in computational complexity.

Chapter 8

The Simplex Minimization Search

8.1 Overview

As already discussed, one of the main requirements for mobile video communication is reduced complexity. In Chapter 7, it was shown that reducing the complexity of the motion estimation process is the best way to reduce the overall complexity of a video codec.

As detailed in Chapter 7 also, there are many techniques for reduced-complexity BMME. The most widely used approach is to use a reduced set of motion vector candidates. Algorithms in this category are usually based on a *unimodal error surface assumption*. In most cases, however, this assumption does not hold true, and such algorithms can easily get trapped in local minima, giving a suboptimal prediction quality. This chapter describes the design of a novel reduced-complexity BMME technique. Although this technique is based on using a reduced set of motion vector candidates, it is designed to be more robust against the local minimum problem.

BMME can be viewed as a *two-dimensional constrained minimization problem*. This problem can, therefore, be solved with reduced-complexity using a wealth of mature optimization techniques. This chapter solves the BMME optimization problem using the *simplex minimization* (SM) optimization method. The resulting solution is called the *simplex minimization search* (SMS). The initialization procedure, termination criterion, and constraints on the independent variables of the search are designed to take into account the basic properties of the BMME problem. This improves the prediction quality of the algorithm and, at the same time, increases its speed-up ratio.

In Chapter 6, it was concluded that one of the main drawbacks of multiple-reference motion-compensated prediction (MR-MCP) is the huge increase in computational complexity. To reduce complexity, this chapter extends the SMS algorithm to the multiple-reference case. Three different novel extensions (or

algorithms) are presented. They represent different degrees of compromise between prediction quality and computational complexity.

The rest of the chapter is organized as follows. Section 8.2 formulates BMME as a two-dimensional constrained optimization problem. The SM method and the reasons for choosing it to solve the BMME problem are described in Section 8.3. The design of the single-reference SMS algorithm is detailed in Section 8.4, and the results of testing it are presented in Section 8.5. Section 8.6 extends the SMS algorithm to the multiple-reference case. The chapter concludes with a discussion in Section 8.7.

Preliminary results of this chapter have appeared in Refs. 165, 166, 167, 168, and 169.

8.2 Block Matching: An Optimization Problem

8.2.1 Problem Formulation

As discussed in Chapter 4 (Section 4.6), in BMME the current frame, f_t, is usually partitioned into nonoverlapping blocks of $N \times N$ pels and the same motion vector is assigned to all pels within a block. The motion vector or displacement, $\mathbf{d} = [d_x, d_y]^T$, of a block is estimated by searching for the best-match block in a larger window of $(N+2d_m) \times (N+2d_m)$ pels centered at the same location in a reference frame, $f_{t-\Delta t}$, where d_m is the maximum allowed motion displacement. This process can be formulated as follows:

$$(d_x, d_y) = \arg\min_{i,j} \mathrm{BDM}(i,j), \tag{8.1}$$

where $-d_m \leq i, j \leq +d_m$ and

$$\mathrm{BDM}(i,j) = \sum_{y=1}^{N} \sum_{x=1}^{N} g[f_t(x,y) - f_{t-\Delta t}(x-i, y-j)]. \tag{8.2}$$

The BDM can be any positive function that measures the distortion between the block in the current frame and the candidate displaced block in the reference frame. Commonly used BDMs are the SSD, $g[\cdot] = (\cdot)^2$, and the SAD, $g[\cdot] = |\cdot|$.

Equations (8.1) and (8.2) clearly indicate that BMME is a *two-dimensional constrained optimization problem*. The two dimensions are the horizontal, i, and vertical, j, motion displacements, the function to be optimized (minimized in this case) is the BDM, and the independent variables, (i,j), are constrained within a limited range, $-d_m \leq i, j \leq +d_m$, and are usually evaluated to a certain accuracy, e.g., full- or half-pel accuracy.

An optimization problem can be thought of as a *search* process where the *function surface* is searched over a given *search space* to find its minimum (or maximum). This search is performed by examining the function value at a finite number of *search locations*. In BMME, the search space is the search window in the reference frame. Each candidate block within this window represents a search location, (i, j). This is the displacement between the block in the current frame and the candidate block in the reference frame. With full-pel accuracy, there are $(2d_m+1)^2$ possible search locations in the search space. The corresponding BDM values form the function surface. Since BDM is a distortion measure, this surface is also referred to as the *error surface*. The set of motion vectors assigned to the blocks of the frame form a *block-motion field*.

8.2.2 A Possible Solution

As shown in Section 8.2.1, BMME can be formulated as an optimization problem. This problem can, therefore, be solved, with reduced complexity, using a wealth of mature optimization methods.

There are few fast BMME algorithms that are based on optimization methods. For example, the TDL search of Jain and Jain [54] is an extension of the *1-D binary logarithmic search* [170], the OTS and CDS algorithms of Srinivasan and Rao [146] are based on the *conjugate directions* (CD) optimization method [171], and the GMS algorithm of Chow and Liu [148] is based on the *genetic algorithm* (GA) optimization method [172].

In a similar fashion, this chapter solves the BMME optimization problem using the *simplex minimization* (SM) optimization method [173]. The resulting solution is called the *simplex minimization search* (SMS).

Figure 8.1 shows the basic building blocks of any constrained optimization method. It can be seen that when trying to solve an optimization problem, there are *two* main design stages. The first, and probably the most important, stage is to choose a suitable optimization method. Section 8.3 describes the SM optimization method and outlines the reasons for choosing it to solve the BMME optimization problem. The second stage is to design a suitable initialization procedure, a termination criterion, and constraints on the independent variables of the search. For the SMS, this stage is detailed in Section 8.4.

8.3 The Simplex Minimization (SM) Optimization Method

8.3.1 Basic Algorithm

Simplex minimization (SM) is a *multidimensional unconstrained* optimization method that was introduced by Nelder and Mead in 1965 [173]. A *simplex* is a

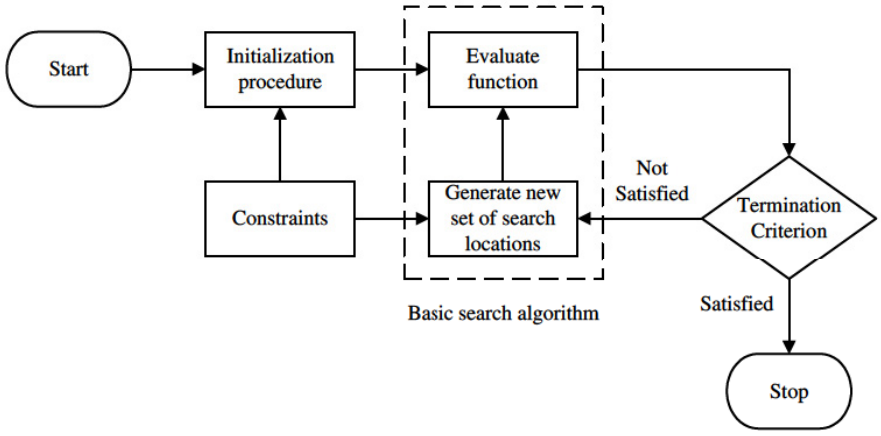


Figure 8.1: Basic building blocks of constrained optimization methods

geometrical figure that consists, in N dimensions, of $N+1$ vertices and all their interconnecting line segments, polygonal faces, etc. Thus, in two dimensions, a simplex is a triangle, whereas in three-dimensions it is a tetrahedron. A *non-degenerate simplex* is one that encloses a finite inner N-dimensional volume.

To minimize a function of N independent variables, the SM method must be initialized with $N+1$ points (or search locations) defining an *initial* nondegenerate simplex. The method then takes a series of steps: *reflecting*, *expanding*, or *contracting* the simplex from the point where the function value is largest, in an attempt to move it to a better point. Thus, the simplex is adapted to the local landscape of the function surface: expanded along inclined planes, reflected on encountering a valley at an angle, and contracted in the neighborhood of a minimum. This process continues until a termination criterion is satisfied. The SM method is described in more detail in Figure 8.2.

8.3.2 Simplex Minimization for BMME: Why?

The SM optimization method is an attractive choice for solving the BMME optimization problem for the following reasons:

1. Most fast BMME algorithms are based on a unimodal error surface assumption. As already shown (Property 4.6.7.3), this assumption does not hold true in most cases. For this reason, such algorithms are easily trapped in local minima, giving a suboptimal prediction quality. The SM method, however, is not based directly on this assumption.

2. Most fast BMME algorithms and optimization methods work by following the direction of the minimum distortion. The SM method,

1. The method is initialized with $N+1$ points, $\mathbf{p}_1, \ldots, \mathbf{p}_{N+1}$, defining an initial nondegenerate simplex where each point is in N dimensions, $\mathbf{p}_i = (p_{i,1}, \ldots, p_{i,N})$. The function to be minimized, f, is then evaluated at those initial vertices to produce the function values $f_1 = f(\mathbf{p}_1), \ldots, f_{N+1} = f(\mathbf{p}_{N+1})$.

2. The highest, $f_h = \max_i f_i$, second highest, $f_s = \max_{i \neq h} f_i$, and lowest, $f_l = \min_i f_i$, function values are determined and the corresponding vertices are marked as $\mathbf{p}_h$, $\mathbf{p}_s$, and $\mathbf{p}_l$, respectively. The centroid, $\mathbf{p}_m$, of the simplex with the highest point removed is then evaluated using

$$\mathbf{p}_m = \frac{1}{N} \sum_{i \neq h} \mathbf{p}_i. \tag{8.3}$$

3. It would seem reasonable to move away from $\mathbf{p}_h$. Thus, the simplex is *reflected* from its highest point about its centroid using

$$\mathbf{p}_r = -\alpha \mathbf{p}_h + (1 + \alpha)\mathbf{p}_m, \tag{8.4}$$

where $\mathbf{p}_r$ is the reflected point and $\alpha \geq 0$ is the *reflection coefficient*. The function is then evaluated at this new reflected point, giving $f_r = f(\mathbf{p}_r)$.

4. IF ($f_r < f_l$), then reflection has produced the lowest function value. Therefore, the direction from $\mathbf{p}_m$ to $\mathbf{p}_r$ seems to be a good one to move along. Thus, the simplex is *expanded* in this direction using

$$\mathbf{p}_e = \gamma \mathbf{p}_r + (1 - \gamma)\mathbf{p}_m, \tag{8.5}$$

where $\mathbf{p}_e$ is the expanded point and $\gamma \geq 1$ is the *expansion coefficient*. The function is then evaluated at this new expanded point, giving $f_e = f(\mathbf{p}_e)$. There are now two possible cases:

   (a) IF ($f_e < f_l$), then the expansion step was in the right direction. Thus, $\mathbf{p}_h$ is removed from the simplex and replaced by $\mathbf{p}_e$. The search then proceeds to step 8 to test for convergence.

   (b) ELSE it seems that the expansion step moved too far in the direction from $\mathbf{p}_m$ to $\mathbf{p}_r$. Thus, $\mathbf{p}_e$ is abandoned. Since $\mathbf{p}_r$ is already known to produce an improvement, $\mathbf{p}_h$ is removed from the simplex and replaced by $\mathbf{p}_r$. The search then proceeds to step 8 to test for convergence.

5. ELSE IF ($f_r > f_l$ AND $f_r < f_s$), then the reflected point is an improvement over the worst two points of the simplex. Thus, $\mathbf{p}_h$ is removed from the simplex and replaced by $\mathbf{p}_r$. The search then proceeds to step 8 to test for convergence.

6. ELSE IF ($f_r > f_i$, for all $i \neq h$), then there are two possible cases:

   (a) IF ($f_r > f_h$), then the search proceeds directly to the contraction step (step 7).

   (b) ELSE $\mathbf{p}_h$ is first removed from the simplex and replaced by $\mathbf{p}_r$ and then the search proceeds to the contraction step (step 7).

Figure 8.2: Simplex method for function minimization

7. It seems that the reflection step moved too far in the direction from $\mathbf{p}_h$ to $\mathbf{p}_m$. This is rectified by *contracting* the simplex from its highest point toward its centroid using

$$\mathbf{p}_c = \beta \mathbf{p}_h + (1 - \beta)\mathbf{p}_m, \tag{8.6}$$

where $\mathbf{p}_c$ is the contracted point and $0 \leq \beta \leq 1$ is the *contraction coefficient*. The function is then evaluated at the new contracted point, giving $f_c = f(\mathbf{p}_c)$. There are now two possible cases:

   (a) IF ($f_c < f_h$), then contraction has produced a better point. Thus, $\mathbf{p}_h$ is removed from the simplex and replaced by $\mathbf{p}_c$. The search then proceeds to step 8 to test for convergence.

   (b) ELSE it would appear that all the efforts to move the highest point to a better location has failed. All the vertices are, therefore, pulled toward the lowest point using

$$\mathbf{p}_i = \frac{\mathbf{p}_i + \mathbf{p}_l}{2}, \quad \text{for all } i \tag{8.7}$$

8. Convergence is tested. IF the convergence criterion is satisfied, then the search is stopped. ELSE the search goes back to step 2.

Figure 8.2: *Continued.*

however, works by moving the point where the function value is largest in different directions using reflection, expansion, and contraction. Thus, it explores directions other than that of the minimum distortion. This makes the method more resilient to the local minimum problem.

3. As shown in Figure 8.1, a very important process in any optimization method is the generation of a new set of search locations for the next iteration. The performance and complexity of any method is highly dependent on this process. The simplest approach is to use a predetermined uniform distribution of search locations. This approach is adopted by most fast BMME algorithms (see the Appendix). There are, however, more complex approaches, like the use of crossover and mutation operators in genetic algorithms or the use of gradients in gradient-descent algorithms. The SM method is a compromise between the two extremes. It uses very simple equations for reflection (8.4), expansion (8.5) and contraction (8.6), as shown in Figure 8.2. As will be shown later, a suitable choice of the coefficients, (α, β, γ), can further reduce the complexity of such equations.

8.4 The Simplex Minimization Search (SMS)

Having decided on the optimization method to be used (the SM optimization method in this case), the second stage is to design a suitable initialization procedure, a termination criterion, and constraints on the independent variables of the search. The performance of an optimization method can be greatly improved if this design stage exploits *a priori* knowledge of the problem at hand. For example, the basic properties of the BMME problem can be exploited to avoid local minima and initialize the search at a location close to the global minimum. This improves the prediction quality and at the same time increases the speed-up ratio.

Although the TDL, OTS, CDS, and GMS algorithms are all based on good optimization methods, they do not take into account the basic properties of the BMME problem. As a result, such algorithms can either get trapped in local minima, resulting in suboptimal prediction quality, or lead to a relatively small speed-up ratio. In the *simplex minimization search* (SMS) algorithm, however, the initialization procedure, termination criterion, and constraints on the independent variables of the search are designed to exploit the basic properties of the BMME problem. This is described in more detail in the following subsections.

8.4.1 Initialization Procedure

Block-matching motion estimation is a two-dimensional problem. As already mentioned, a simplex, in two-dimensions, is a *triangle*. Thus, *three* points need to be chosen to define the initial nondegenerate simplex. As is shown later, the performance of the SM method is highly dependent on the choice of these points. The following initialization procedure is used.

According to Property 4.6.7.1, the vector $(0,0)$ has the highest probability of occurrence within the block-motion field. One of the initial three points is therefore set to $(0,0)$. In addition, Property 4.6.7.2 states that there is a high correlation between the motion vectors of adjacent blocks. In fact, most video coding standards take advantage of this property by predictively coding the motion vectors. To exploit this property, and to match the motion estimation process to the motion coding process, the other two points of the initial simplex are set to the motion vectors of the blocks *above* and to the *left* of the current block. If such neighboring vectors are not available, as in border blocks, they are set to $(0,0)$.

Note that this procedure does not guarantee to produce a nondegenerate initial simplex. For example, if two points are identical, then the simplex is degenerate. In this case, a *local* search is applied to find other candidates. The BDM is first evaluated at the points chosen by the foregoing procedure.

Let $\mathbf{p}_m = (m_x, m_y)$ be the point that yields the smallest BDM, then the BDM is also evaluated at its eight nearest neighbors, $(m_x, m_y \pm a)$, $(m_x \pm a, m_y)$ and $(m_x \pm a, m_y \pm a)$, where a is the accuracy of the search, e.g., $a = 1$ for full-pel accuracy. At this stage, all points (including those from the initial procedure) are arranged in ascending order according to their BDMs and the first three are chosen to form the initial simplex. If this is still a degenerate one, then the appropriate point is dropped and replaced by the next one in the list. This is repeated until a nondegenerate simplex is formed.

Once a nondegenerate initial simplex is formed, the search proceeds as shown in Figure 8.2, subject to the constraints outlined in Section 8.4.2, and is terminated when the criterion described in Section 8.4.3 is satisfied.

8.4.2 Constraints on the Independent Variables

The SM method assumes continuous unconstrained independent variables. However, when applied to the constrained minimization problem of BMME, two constraints have to be imposed. Firstly, the vertices of the simplex must always be set to the required accuracy before any BDM evaluation can take place. For example, if full-pel accuracy is assumed, then any point produced by reflection, expansion, or contraction must be rounded to the nearest integer value. Secondly, the vertices of the simplex must always be kept within the search window. Any point produced by reflection, expansion, or contraction must be set to the closest point within the range $-d_m \leq i, j \leq +d_m$ before any BDM evaluation can take place. This constraint is more efficient than other possible constraints, like, for example, assigning a large function value to the vertex outside the search window.

8.4.3 Termination Criterion

There are many possible ways to terminate optimization methods. One of the most widely used approaches is to terminate the search if the current minimum function value is below some threshold. In the SM case, another approach is to terminate the search if the fractional range from the highest, in terms of function value, to the lowest vertices of the simplex is below some threshold [174].

According to Property 4.6.7.4, the function value at the global minimum is unpredictable. Thus, if the preceding termination criteria are used, then the threshold needs to be adjusted from one sequence to another, from one frame to another, and even from one block to another. This makes such criteria unsuitable for BMME. A more suitable criterion is as follows. Let $\mathbf{p}_h = (h_x, h_y)$, $\mathbf{p}_s = (s_x, s_y)$, and $\mathbf{p}_l = (l_x, l_y)$ be the vertices of the simplex where the BDM is

highest, second highest, and lowest, respectively. The search is terminated if the following condition is satisfied:

$$(|h_x - l_x| \leq a) \wedge (|h_y - l_y| \leq a) \wedge (|s_x - l_x| \leq a) \wedge (|s_y - l_y| \leq a), \qquad (8.8)$$

where a is the search accuracy and $\wedge$ is the logical AND operator. In other words, the search is terminated *if the two highest (in terms of BDM value) vertices of the simplex become neighbors to the lowest vertex*. This criterion was derived from the way the SM method works. As shown in Figure 8.2, when the method converges to a minimum, the contraction operation starts pulling all the vertices toward the minimum vertex. The main advantage of this criterion is that it does not depend on a threshold.

8.4.4 Motion Vector Refinement

The main disadvantage of the preceding termination criterion is that it is not based directly on the function to be minimized, i.e., the BDM. As a result, the search may sometimes converge to a suboptimal point. Experimental results show that in most cases this suboptimal point is in the neighborhood of the global minimum. An extra step is therefore added to the search in which the motion vector produced by SM is *refined* by searching its eight nearest neighbors. Note that this does not significantly increase the complexity of the search, because most of those neighbors have already been searched.

8.5 Simulation Results

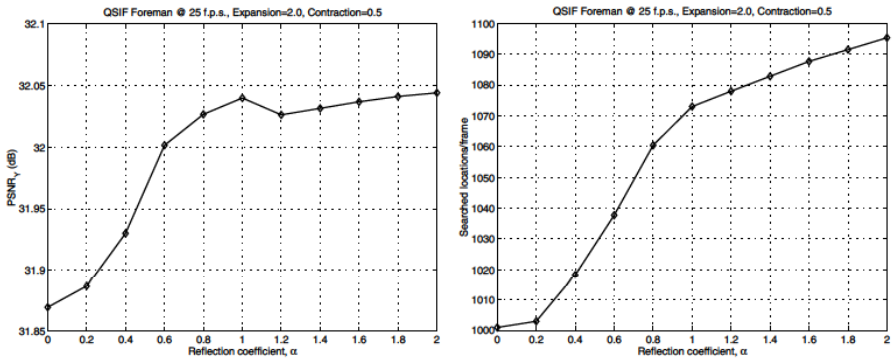
8.5.1 Results Within an Isolated Test Environment

In this set of simulations, motion is estimated and compensated using original reference frames. In effect, this is equivalent to *lossless* DFD coding. This is particularly important for a fair comparison between different algorithms on a frame-by-frame basis, since poor prediction of one frame does not propagate to, and affect the prediction of, the next frame. Hereafter, the term *isolated test environment* will be used to refer to this test condition.

All results in this subsection were generated using blocks of 16×16 pels, a maximum allowed displacement of ± 15 pels, SAD as the distortion measure, restricted motion vectors, and full-pel accuracy. Motion vectors were coded predictively using the prediction method and the VLC table of the H.263 standard. All quoted results refer to the luma components of sequences.

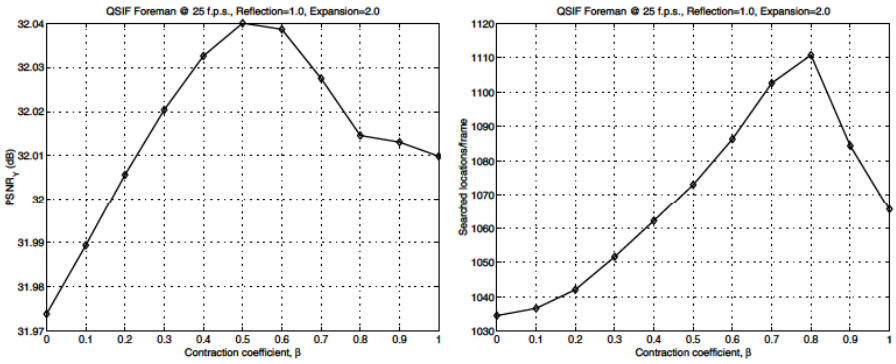
8.5.1.1 Choice of Coefficients

Before evaluating the performance of the SMS algorithm, suitable values for the reflection, α, contraction, β, and expansion, γ, coefficients need to be chosen. Figures 8.3, 8.4, and 8.5 show the performance of the SMS algorithm with different values of α, β, and γ, respectively. The figures indicate that the performance of the SMS algorithm is not very sensitive to the choice of these coefficients. This may be due to the good performance of the initialization procedure and termination criterion. In general, however, the values $\alpha = 1$, $\beta = \frac{1}{2}$, and $\gamma = 2$ provide the best compromise between computational complexity and prediction quality. In addition, this particular set of coefficients reduces the complexity of the SM transformation equations, Equations (8.4),



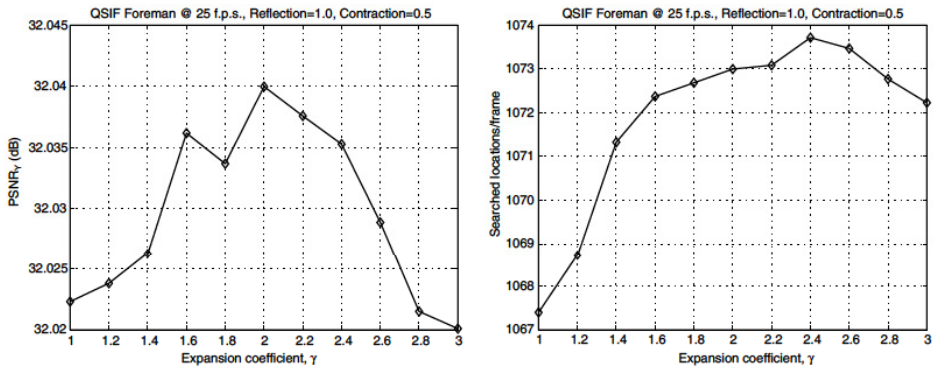
(a) Prediction quality (b) Computational complexity

Figure 8.3: Performance of SMS with different values of the reflection coefficient α



(a) Prediction quality (b) Computational complexity

Figure 8.4: Performance of SMS with different values of the contraction coefficient β



(a) Prediction quality (b) Computational complexity

Figure 8.5: Performance of SMS with different values of the expansion coefficient γ

(8.5), and (8.6) in Figure 8.2, because multiplications and divisions in this case can be performed using shift operations.

8.5.1.2 Initialization, Termination, and Refinement Tests

In order to justify different parts of the SMS algorithm, the following tests were performed.

1. **Initialization Test**: Two initialization procedures were tested:
   (a) *Random Initialization*: Two of the vertices of the initial simplex are generated randomly within the search window, whereas the third vertex is always set to $(0, 0)$.
   (b) *Proposed Initialization*: This is the initialization procedure described in Section 8.4.1.
2. **Termination Test**: Two termination criteria were tested.
   (a) *Threshold Termination*: The search is terminated when the current minimum BDM value is below a threshold. The threshold was set to 768, which corresponds to an average SAD/pel of 3 ($16 \times 16 \times 3$). As already discussed, a fixed threshold is not suitable for the BMME problem. Such a threshold does not guarantee convergence because the global minimum BDM value may in some cases be above the threshold. The threshold condition must therefore be supported by another condition to guarantee termination. In this test, the search is also terminated if the number of iterations exceeds 10.
   (b) *Proposed Termination*: This is the termination criterion described in Section 8.4.3.

Table 8.1: Initialization, termination, and refinement tests

	Akiyo		Foreman		Table Tennis	
	PSNR (dB)	Locations	PSNR (dB)	Locations	PSNR (dB)	Locations
Initialization test:						
Random	45.93	1,634	31.32	1,890	31.28	1,647
Proposed	45.93	684	32.04	1,073	31.71	831
Termination test:						
Threshold	45.93	904	32.06	1,396	31.73	1,082
Proposed	45.93	684	32.04	1,073	31.71	831
Refinement test:						
No refinement	45.93	683	31.97	999	31.53	794
Proposed	45.93	684	32.04	1,073	31.71	831

3. **Refinement Test**: Two cases were tested.

   (a) *Proposed Refinement*: The motion vector produced by SM is refined by searching its eight nearest neighbors, as described in Section 8.4.4.

   (b) *No Refinement*: No refinement is performed.

Table 8.1 summarizes the results of the preceding tests. The results are averaged over each sequence with a frame skip of 1. Prediction quality is given in terms of average luma PSNR (dB), and computational complexity is given in terms of average searched locations per frame. The results clearly justify the use of the proposed initialization procedure, termination criterion, and refinement step.

8.5.1.3 Performance Evaluation

In addition to the SMS algorithm, five BMME algorithms were simulated: the full-search (FS) algorithm, the two-dimensional logarithmic search (TDL) [54], the cross-search algorithm (CSA) [147], the one-at-a-time search (OTS) [146], and the N-steps search (NSS), which is the general form[1] of the three-steps search (TSS) [145]. In this case the number of steps in the NSS search is set to $N = 4$ to give a maximum displacement of ± 15 pels. A detailed description of these fast BMME algorithms is given in the Appendix.

[1]The three-steps search starts with ± 4 pels in the first step, then ± 2 pels in the second step, and ± 1 pel in the third step. This gives a maximum allowed displacement of $\pm 4 \pm 2 \pm 1 = \pm 7$ pels. For larger search windows the number of steps must be increased. This is called the N-steps search. For example, when $N = 4$, the search has 4 steps and the first step starts with ± 8 pels, giving a maximum allowed displacement of ± 15 pels.

Table 8.2: Comparison between different block-matching algorithms in terms of prediction quality

	Akiyo			Foreman			Table Tennis		
	PSNR	ΔPSNR	% Global	PSNR	ΔPSNR	% Global	PSNR	ΔPSNR	% Global
FS	45.93	0.00	100.00	32.20	0.00	100.00	32.17	0.00	100.00
SMS	**45.93**	**0.00**	**100.00**	**32.04**	**−0.16**	**94.31**	**31.71**	**−0.46**	**95.80**
NSS	45.93	0.00	100.00	31.74	−0.46	87.25	31.50	−0.67	92.74
TDL	45.93	0.00	100.00	31.81	−0.39	88.92	31.63	−0.54	93.39
CSA	45.91	−0.02	99.86	30.95	−1.25	60.11	30.93	−1.24	81.23
OTS	45.93	0.00	100.00	31.23	−0.97	76.35	31.23	−0.94	91.60

Table 8.3: Comparison between different blockmatching algorithms in terms of computational complexity

	Akiyo		Foreman		Table Tennis	
	Locations	Speed-up	Locations	Speed-up	Locations	Speed-up
FS	65,621	–	77,439	–	65,621	–
SMS	**684**	**96**	**1,073**	**72**	**831**	**79**
NSS	2,464	27	2,823	27	2,473	27
TDL	1,310	50	1,638	47	1,362	48
CSA	115	571	920	84	461	142
OTS	402	163	604	128	448	146

Tables 8.2, 8.3, and 8.4 compare the performance of the simulated BMME algorithms. All results are averages over sequences with a frame skip of 1. Table 8.2 compares the prediction quality in terms of average luma PSNR in decibels. The difference in PSNR between each algorithm and the FS algorithm is also shown.[2] The table also shows the average percentage of finding the global minimum. Table 8.3, on the other hand, compares the computational complexity in terms of average searched locations per frame. It also shows the speed-up ratio[3] of each algorithm with reference to the FS algorithm. Table 8.4 shows the motion overhead generated by each algorithm and the difference between this overhead and that produced by the FS algorithm.[4]

As expected, the FS algorithm provides the best prediction quality, but at the expense of a very high computational complexity. The fast BMME algorithms in this simulation can be split into three difference performance classes. In the first class, the CSA and the OTS algorithms provide the highest

[2] ΔPSNR = PSNR of fast algorithm − PSNR of FS algorithm.

[3] $\text{Speed-up} = \dfrac{\text{Searched locations for FS algorithm}}{\text{Searched locations for fast algorithm}}$.

[4] ΔBits = Motion bits of fast algorithm − Motion bits of FS algorithm.

Table 8.4: Comparison between different block-matching algorithms in terms of motion overhead

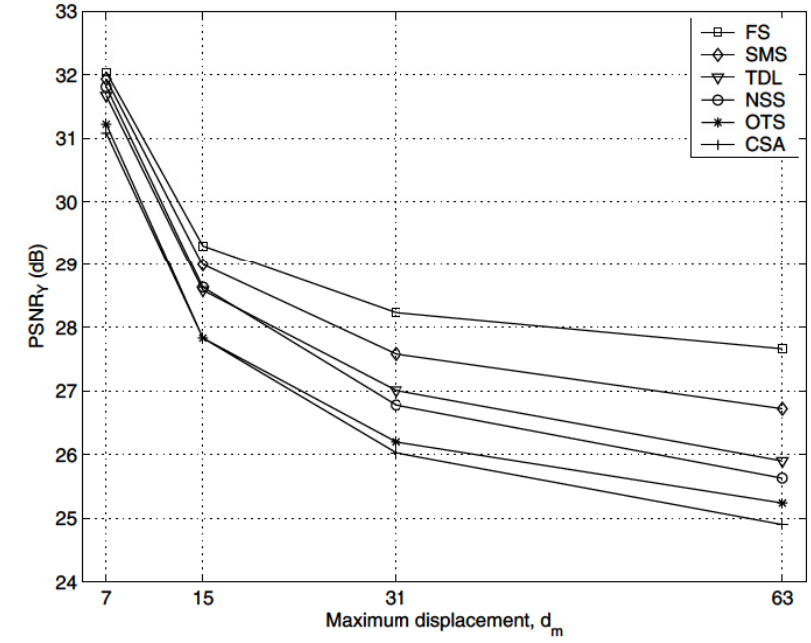
	Akiyo		Foreman		Table Tennis	
	Motion bits	ΔBits	Motion bits	ΔBits	Motion bits	ΔBits
FS	177	0	388	0	279	0
SMS	**177**	**0**	**358**	**−30**	**247**	**−32**
NSS	177	0	457	+69	290	+11
TDL	177	0	394	+6	269	−10
CSA	177	0	461	+73	281	+2
OTS	177	0	388	0	246	−33

speed-up ratios, but their prediction quality deteriorates for sequences with medium to high movement content. In the second class, the NSS and the TDL algorithms provide better prediction quality than CSA and OTS, but at the expense of a higher computational complexity. In the third class, the SMS algorithm provides the best compromise between prediction quality and computational complexity. Its prediction quality is the closest to that of FS and yet its computational complexity is between those of the other two classes. Note that the SMS algorithm achieves the highest percentage of finding the global minimum. This clearly indicates that the SMS algorithm is the most resilient to the local minimum problem. Note also that the SMS algorithm adapts better to the movement content of sequences. Thus, for low-movement sequences it uses fewer locations and for high-movement sequences it uses more locations. In addition, because the motion estimation process is matched to the motion coding process (through the initialization procedure), the SMS algorithm has the lowest motion overhead.

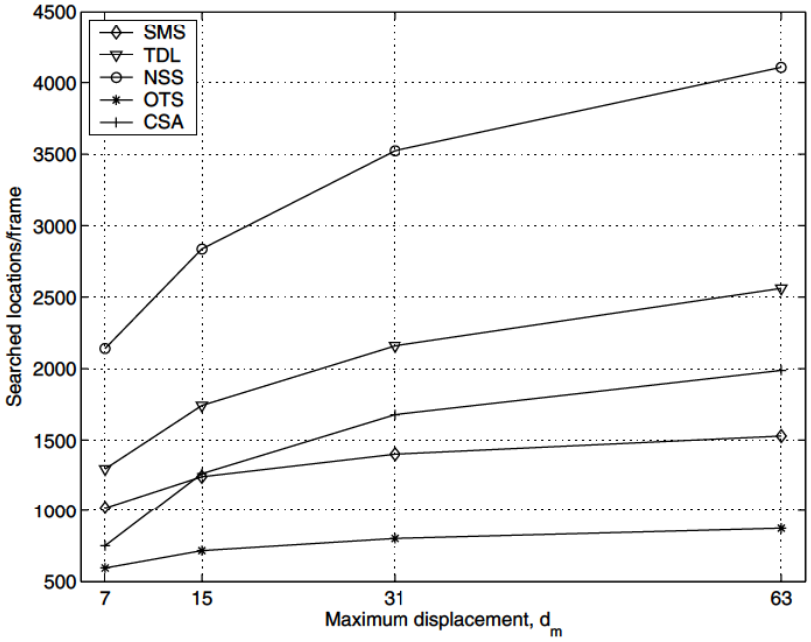
One of the disadvantages of fast BMME algorithms is that their prediction quality deteriorates for higher amounts of motion and larger search windows (as, for example, in HDTV applications). This is clear from Table 8.2 when moving from Akiyo to Foreman and Table Tennis. To investigate this effect further, the Foreman sequence was temporally subsampled to 25, 12.5, 8.33, and 6.25 frames/s (this corresponds to frame skips of 1, 2, 3, and 4, respectively). The corresponding maximum allowed displacements, d_m, were set to ± 7, ± 15, ± 31, and ± 63 pels, respectively. Figure 8.6 shows the results of this simulation. It is immediately evident that the SMS algorithm is the most robust fast algorithm to the above effect, and yet it has the second-lowest computational complexity.

8.5.2 Results Within an H.263-like Codec

The SMS algorithm along with the other five BMME algorithms have also been tested within a hybrid H.263-like codec. As in previous simulations,



(a) Prediction quality



(b) Computational complexity

Figure 8.6: Comparison between different block-matching algorithms when applied to QSIF FOREMAN with maximum displacements of $7, 15, 31$, and 63 and corresponding frame rates of $25, 12.5, 8.33$, and 6.25 frames/s, respectively

motion was estimated using macroblocks of 16×16 pels, a maximum allowed displacement of ± 15 pels, SAD as the distortion measure, restricted motion vectors, and full-pel accuracy. In this case, however, motion vectors were predictively encoded using the median prediction method and the VLC table of the H.263 standard. In addition, motion was estimated and compensated using reconstructed reference frames rather than original frames. Both, the frame signal (in case of INTRA) and the DFD signal (in case of INTER) were transform encoded according to the H.263 standard. To generate a range of bit rates, the quantization parameter QP was varied over the range 5–30 in steps of 5. This means that each algorithm was used to encode a given sequence six times. Each time, QP was held constant over the whole sequence (i.e., no rate control was used). The first frame was always INTRA encoded, and all other frames were INTER encoded. No INTRA/INTER switching was allowed at the macroblock level. The INTRA bits were included in the bit-rate calculations, and no header bits were generated. All quoted results refer to the luma components of sequences.

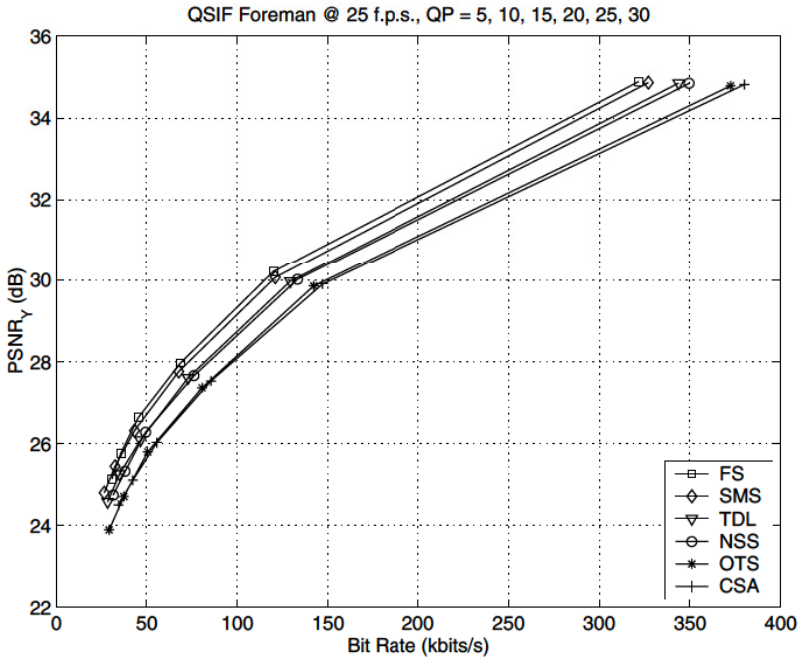
Figures 8.7 and 8.8 show examples of the rate-distortion (R-D) performance of the SMS algorithm and compare it to that of the other five BMME algorithms. Figure 8.7 shows the results for the FOREMAN sequence with frame rates of 25 frames/s and 8.33 frames/s, whereas Figure 8.8 shows the results for the AKIYO and TABLE TENNIS sequences with frame rates of 10 frames/s and 15 frames/s, respectively. Both figures confirm the superior R-D performance of the SMS algorithm compared to other fast BMME algorithms.

The superior performance of the SMS algorithm is also shown on a frame-by-frame basis in Figure 8.9. This figure shows the performance for the FOREMAN sequence at 8.33 frames/s with a quantization parameter of $QP = 10$. For clarity, the figure shows only the performance of the FS, SMS, NSS, and OTS algorithms. As can be seen, the SMS algorithm provides the closest prediction quality (Figure 8.9(a)) to the FS algorithm. This results in the use of fewer bits for the DFD signal (Figure 8.9(c)). In addition, the initialization procedure results in less motion overhead (Figure 8.9(d)). The reduced number of DFD bits and motion bits results in a reduced overall bit rate (Figure 8.9(e)). This is all achieved at a reduced computational complexity (Figure 8.9(b)).

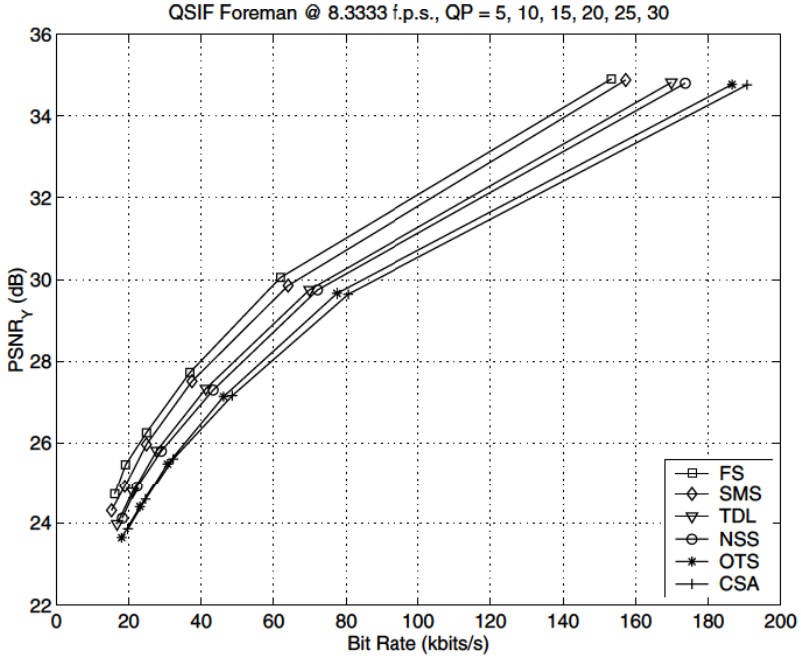
8.5.3 Results Within an MPEG-4 Codec

In a collaborative work, the SMS algorithm has also been tested within an MPEG-4 codec. The results in this subsection are reproduced, as is, from Ref. 175.[5]

[5]The authors would like to thank Mr. Oliver Sohm for incorporating SMS within MPEG-4 and providing the results.

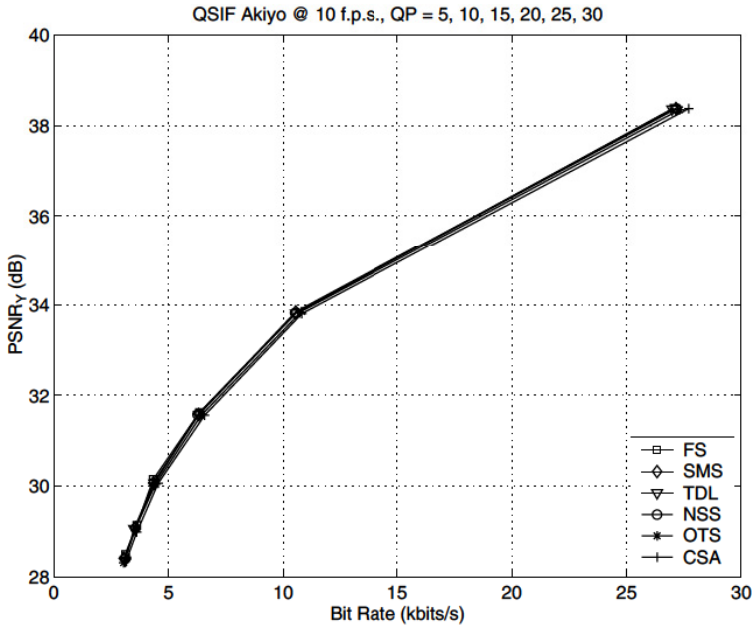


(a) FOREMAN at 25 frames/s (skip = 1)

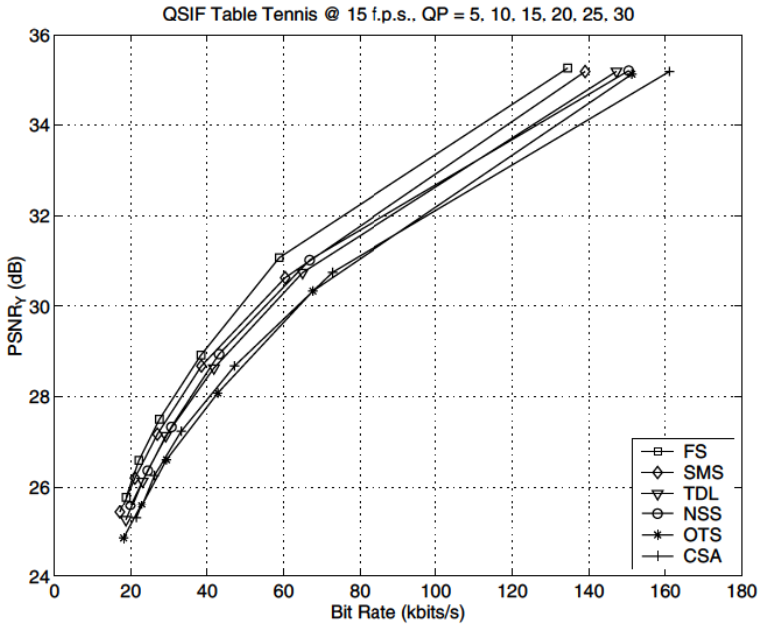


(b) FOREMAN at 8.3 frames/s (skip = 3)

Figure 8.7: R-D performance of different block-matching algorithms when applied to QSIF FOREMAN



(a) AKIYO at 10 frames/s (skip = 3)



(b) TABLE TENNIS at 15 frames/s (skip = 2)

Figure 8.8: R-D performance of different block-matching algorithms when applied to QSIF AKIYO and QSIF TABLE TENNIS

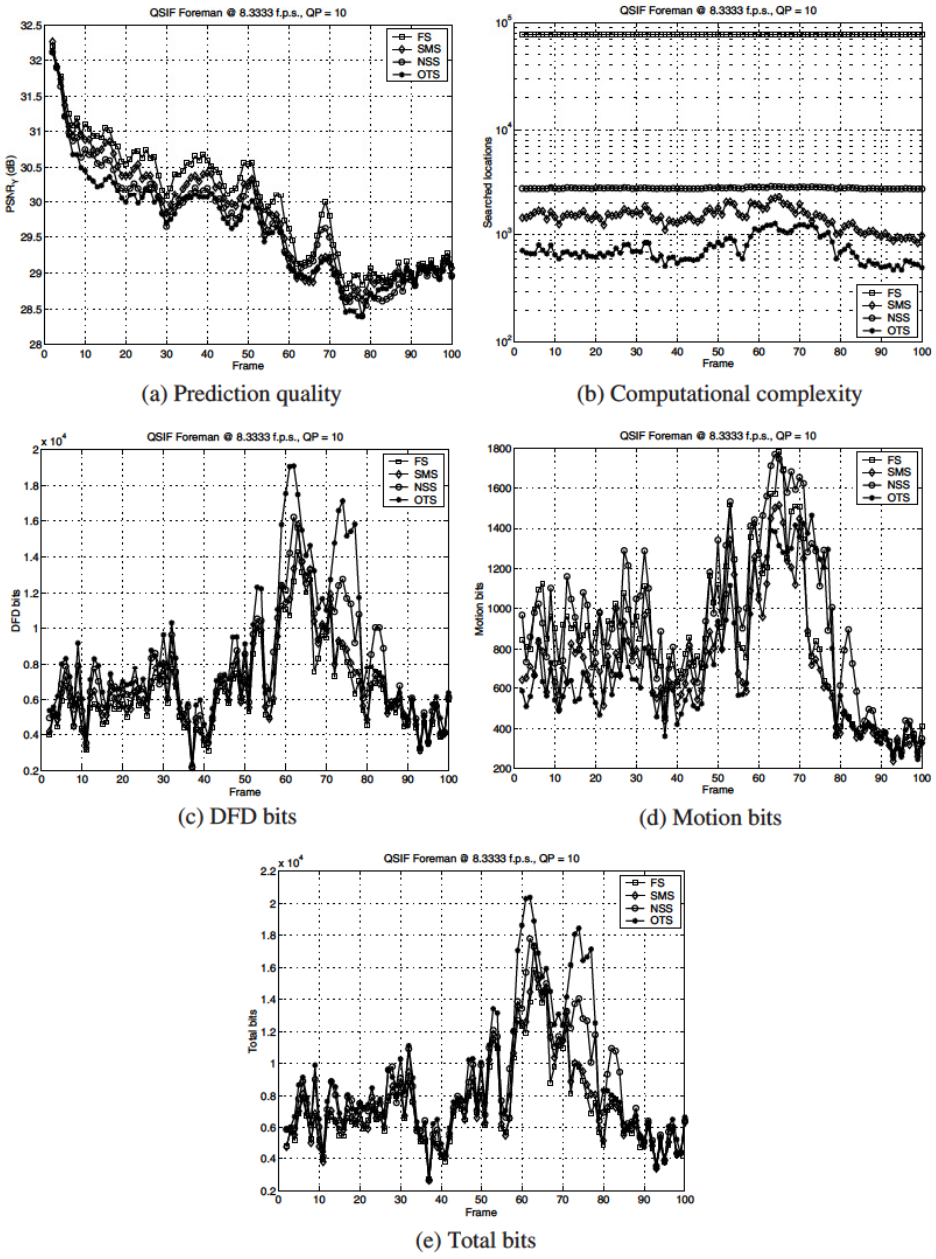


Figure 8.9: Comparison between different block-matching algorithms when applied to QSIF FOREMAN at 8.33 frames/s and QP = 10

They are provided here to show the performance of the SMS algorithm within an object-based video codec.

Before proceeding to present the results, a description of object-based motion estimation in the MPEG-4 verification model [176] is in order. To account for arbitrarily shaped objects, the standard block-matching algorithm is extended to *polygon matching*. Macroblock-based repetitive padding is used for the reference visual object plane (VOP). In other words, macroblocks that lie on the VOP boundary are padded so that pels from inside the VOP are extrapolated to the outside. For each 16×16 macroblock in the current VOP, full-pel full search is used to find the motion vector that minimizes the SAD. The SAD of the motion vector $(0,0)$ is reduced by a preset threshold to favor this vector. A reduced search of ± 2 pels centered around the 16×16 motion vector is used to find one motion vector for each of the four 8×8 blocks within the MB. A decision is then made whether to use one motion vector or four motion vectors per MB. A decision is also made whether to encode the MB in INTRA or INTER mode. If INTER mode is chosen, the 16×16 (or the four 8×8) vector(s) is/are refined to half-pel accuracy using a reduced $\pm 1/2$-pel search centered around the full-pel vector. Motion vectors are restricted within the bounding box of the VOP unless the unrestricted mode is chosen. In this mode, the reference VOP is extended by repetitive padding in all directions by the number of pels which equals the search range. Overlapped motion compensation is similar to that of H.263.

In this set of simulations, four algorithms were tested: FS, SMS, NSS, and diamond search (DS) [149, 150, 151] (which is adopted in the MPEG-4 verification model [176]). To ensure that the global minimum is found, the threshold that favors the $(0,0)$ vector in the FS algorithm was set to zero. The four algorithms were used only for the full-pel search. All other operations (e.g., 8×8 ME, half-pel refinement) remained the same. Original reference VOPs were used instead of reconstructed VOPs. The unrestricted motion vector mode was switched on. Table 8.5 gives more details about the test conditions and the test sequences.

Table 8.6 shows the prediction quality in terms of mean absolute error per pel (MAE/pel),[6] whereas Table 8.7 shows the computational complexity in terms of average searched locations per macroblock (locations/MB). Again, the superior performance of the SMS algorithm is evident. Compared to NSS and DS, the SMS algorithm provides the closest MAE/pel to that of FS, and yet it has the least number of searched locations/MB.

[6]The MAE/pel measure was calculated as follows. The minimum SADs over the whole VOP were summed and then divided by the number of opaque pels in the VOP. The minimum SADs in this case are those produced by the full-pel search.

Table 8.5: Test sequences and conditions for the MPEG-4 results. Reproduced from Ref. 175

Sequence	Format	Class	Objects	Distance	Displacement
Bream	CIF, 352 × 288, 30 Hz, 300 frames	E	VO0: Background VO1: Fish	2 (15 f.p.s.)	−16 ... 15
Coast Guard	QCIF, 176 × 144, 30 Hz, 270 frames	B	VO0: Water VO1: Small boat VO2: Big boat VO3: River bank	3 (10 f.p.s.)	−16 ... 15
Container ship	SIF, 352 × 240, 30 Hz, 161 frames	A	VO0: Water VO1: Ship VO2: Small boat VO3: Land (fg) VO4: Sky+Land (bg) VO5: Flag	4 (7.5 f.p.s.)	−16 ... 15
News	QCIF, 176 × 144, 30 Hz, 300 frames	B	VO0: Background VO1: Dancers VO2: News readers VO3: Text	3 (10 f.p.s.)	−16 ... 15
Stefan	CIF, 352 × 288, 30 Hz, 300 frames	C	VO0: Stefan	1 (30 f.p.s.)	−16 ... 15

Table 8.6: Prediction quality within MPEG-4 in terms of MAE/pel. Reproduced from Ref. 175

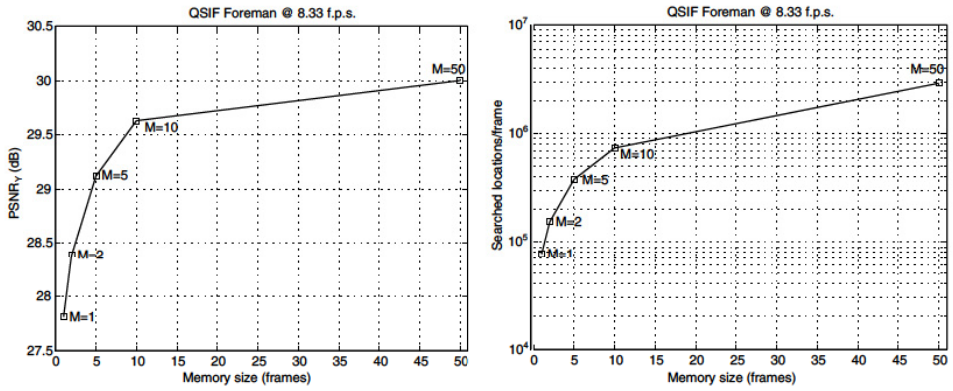
Sequence	Object	FS	SMS	DS	NSS
Bream	VO1: Fish	6.277	6.533	8.416	9.709
Coast Guard	VO0: Water	3.797	3.847	3.993	4.014
	VO1: Small boat	5.197	5.374	5.692	5.543
	VO2: Big boat	4.696	4.898	5.096	5.071
	VO3: River bank	4.591	4.636	4.885	6.523
Container ship	VO0: Water	2.287	2.288	2.357	2.358
	VO1: Ship	2.069	2.069	2.082	2.113
	VO2: Small boat	2.154	2.159	2.166	2.181
	VO3: Land (fg)	0.831	0.831	0.831	0.831
	VO4: Sky+Land (bg)	0.792	0.801	0.839	0.843
	VO5: Flag	15.828	16.066	16.105	16.131
News	VO0: Background	0.060	0.061	0.061	0.061
	VO1: Dancers	5.568	5.773	5.860	5.852
	VO2: News readers	1.153	1.154	1.159	1.158
	VO3: Text	0.092	0.092	0.092	0.092
Stefan	VO0: Stefan	8.200	8.662	9.346	9.430
Average		3.975	4.078	4.311	4.494
Relative to FS		100.0%	102.6%	108.5%	113.1%

Table 8.7: Computational complexity within MPEG-4 in terms of searched locations/MB. Reproduced from Ref. 175

Sequence	Object	FS	SMS	DS	NSS
Bream	VO1: Fish	1,017.2	16.7	21.4	33.0
Coast Guard	VO0: Water	1,021.8	16.4	18.0	33.0
	VO1: Small boat	1,004.7	16.8	17.5	32.9
	VO2: Big boat	1,010.0	17.7	19.1	33.0
	VO3: River bank	1,020.5	13.1	19.2	32.9
Container ship	VO0: Water	1,023.4	12.7	13.1	33.0
	VO1: Ship	1,023.7	11.4	13.5	33.0
	VO2: Small boat	1,014.2	14.7	15.9	32.9
	VO3: Land (fg)	1,024.0	9.4	13.0	33.0
	VO4: Sky+Land (bg)	1,024.0	13.9	13.1	33.0
	VO5: Flag	1,012.3	19.1	15.6	33.0
News	VO0: Background	1,024.0	9.3	13.0	33.0
	VO1: Dancers	1,024.0	15.4	16.3	33.0
	VO2: News readers	1,022.9	9.8	13.1	33.0
	VO3: Text	1,024.0	9.0	13.0	33.1
Stefan	VO0: Stefan	1,002.4	21.8	22.2	32.9
Minimum		1,002.4	9.0	13.0	32.9
Maximum		1,024.0	21.8	22.2	33.1
Average		1,018.3	14.2	16.1	33.0

8.6 Simplex Minimization for Multiple-Reference Motion Estimation

As already discussed, MR-MCP achieves significant prediction gains, but at the expense of a significant increase in computational complexity. This is illustrated in Figure 8.10 for the FOREMAN sequence at 8.33frames/s. This figure was generated using the same simulation conditions described in Section 6.3.2. Figure 8.10(a) shows the prediction quality (in terms of PSNR_Y in decibels) as a function of multiframe memory size (in frames), whereas Figure 8.10(b) shows the computational complexity (in terms of searched locations/frame). It is clear that increasing the memory size M increases the prediction quality. This is, however, at the expense of a linear increase in computational complexity. The aim of this section is to design fast long-term memory block-matching algorithms that can reduce computational complexity but at the same time maintain the prediction gain of multiple-reference motion estimation.



(a) Prediction quality (b) Computational complexity

Figure 8.10: Performance of LTM-MCP as a function of memory size for QSIF FOREMAN at 8.33 frames/s

8.6.1 Multiple-Reference SMS Algorithms

This section extends the SMS algorithm to the multiple-reference case. As detailed in Section 8.4, the design of the SMS algorithm was based on some important properties of the block-motion fields of typical video sequences. In particular, the design was based on Properties 4.6.7.1 and 4.6.7.2 of the single-reference block-motion field. The two properties are the center-biased distribution of the field and the high correlation between adjacent motion vectors, respectively. The results of the investigation in Section 6.3.1 indicate that the two properties still hold true in the multiple-reference case (Properties 6.3.1.1 and 6.3.1.3). Thus, the efficient performance of the SMS algorithm can be extended to the multiple-reference case without the need for a major redesign. Three different extensions (or algorithms) are described in what follows.

MR-SMS This is a direct extension of SMS. For each block in the current frame, the single-reference SMS algorithm is used to individually search each frame in the multiframe memory and produce a best-match block from that frame. The overall best-match is then chosen from this set of M blocks.

MR-FS/SMS This is the same as MR-SMS, but the most recent reference frame in memory (i.e., the frame for which $d_t = 0$) is searched using full search instead of SMS. Giving more importance to searching this frame is motivated by Property 6.3.1.2, which states that the most recent reference frame has the highest probability of selection.

MR-3DSM The single-reference SMS algorithm is based on a two-dimensional version of the simplex minimization (SM) optimization method

(Section 8.3). Algorithm MR-3DSM, however, is based on a *three-dimensional* version ($N = 3$ in Figure 8.2). A 3-D version of SM must be initialized with *four* locations defining an initial simplex in the search space. For the MR-3DSM algorithm, this is achieved as follows. For each block in the current frame, the initialization procedure described in Section 8.4.1 is applied individually to each frame in the multiframe memory. This will generate three initial vertices from each frame. The best *four* vertices, in terms of BDM value, are selected from this set of $3M$ vertices. A procedure similar to that described in Section 8.4.1 is used to ensure that the four vertices form a nondegenerate simplex. This simplex is used to initialize the 3-D version of SM, where the third dimension here is the temporal displacement. The same criterion described in Section 8.4.3 is used to terminate the algorithm, with the added condition that *the four vertices of the final simplex must have the same temporal displacement*.

8.6.2 Simulation Results

The multiple-reference SMS algorithms were tested using the luma components of the three QSIF sequences AKIYO, FOREMAN, and TABLE TENNIS with full-pel accuracy, blocks of 16×16 pels, a maximum allowed displacement of ± 15 pels, SAD as the distortion measure, restricted motion vectors, and original reference frames. In addition to the multiple-reference SMS algorithms, the single-reference full-search (SR-FS) and the multiple-reference full-search (MR-FS) algorithms were also simulated. For the multiple-reference algorithms, sliding-window control was used to maintain a long-term memory of size $M = 50$ frames.

Tables 8.8 and 8.9 compare the performance of the simulated algorithms. All results are averages over sequences with a frame skip of 1. Table 8.8 compares the prediction quality in terms of average luma PSNR in decibels. The difference[7] in PSNR between each algorithm and the MR-FS algorithm is also shown. Table 8.9, on the other hand, compares the computational complexity in terms of average searched locations per frame. It also shows the speed-up ratio[8] of each algorithm with reference to the MR-FS algorithm.

It is immediately evident that the multiple-reference SMS algorithms provide significant reductions in computational complexity compared to the MR-FS algorithm. The SMS algorithms represent different degrees of compromise between prediction quality and computational complexity. At one extreme is

[7] $\Delta\text{PSNR} = \text{PSNR of fast algorithm} - \text{PSNR of MR-FS algorithm}$.

[8] $\text{Speed-up} = \dfrac{\text{Searched locations for MR-FS algorithm}}{\text{Searched locations for fast algorithm}}$.

Table 8.8: Comparison between different block-matching algorithms in terms of prediction quality (average PSNR_Y in dB) with a multiframe memory of $M = 50$ frames and a frame skip of 1

	AKIYO		FOREMAN		TABLE TENNIS	
	PSNR	ΔPSNR	PSNR	ΔPSNR	PSNR	ΔPSNR
SR-FS	45.93	−0.62	32.20	−1.77	32.17	−0.70
MR-FS	46.55	0.00	33.97	0.00	32.87	0.00
MR-FS/SMS	46.55	0.00	33.92	−0.05	32.80	−0.07
MR-SMS	46.55	0.00	33.87	−0.10	32.67	−0.20
MR-3DSM	46.55	0.00	33.51	−0.46	32.46	−0.41

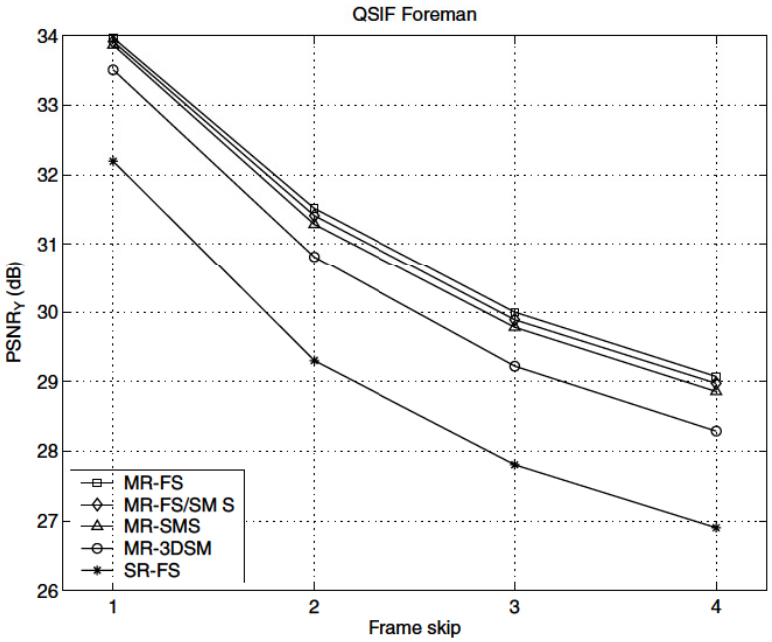
Table 8.9: Comparison between different block-matching algorithms in terms of computational complexity (average searched locations/frame) with a multiframe memory of size $M = 50$ frames and a frame skip of 1

	AKIYO		FOREMAN		TABLE TENNIS	
	Locations	Speed-up	Locations	Speed-up	Locations	Speed-up
SR-FS	65,621	45.90	77,439	45.90	65,621	45.90
MR-FS	3,012,200	1.00	3,554,700	1.00	3,012,200	1.00
MR-FS/SMS	103,820	29.01	183,240	19.40	134,270	22.43
MR-SMS	38,880	77.47	106,830	33.27	69,443	43.38
MR-3DSM	35,867	83.98	66,357	53.57	45,518	66.18

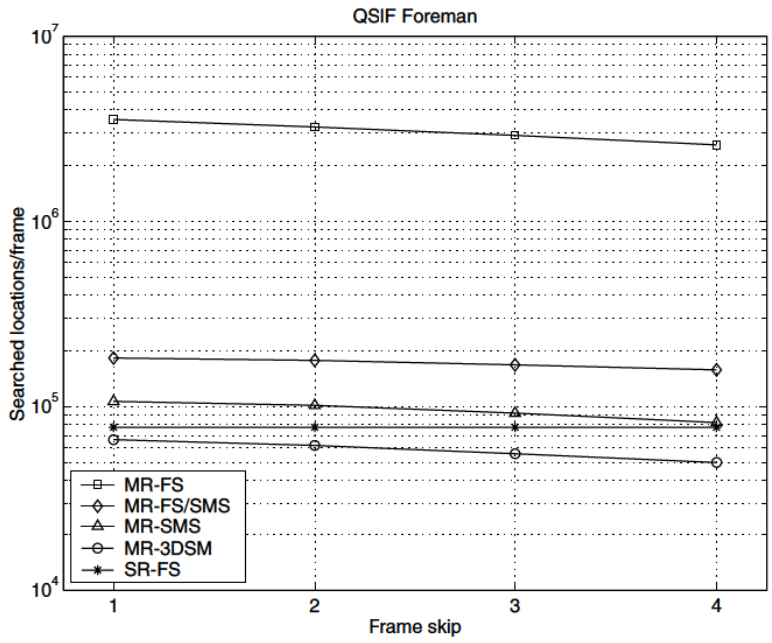
the MR-3DSM algorithm. Compared to MR-FS, the MR-3DSM algorithm provides significant reductions in computational complexity (a speed-up ratio of about 54–84) at the expense of a moderate reduction in prediction quality (about 0.41–0.46 dB loss[9]). At the other extreme is the MR-FS/SMS algorithm. It uses full search on the most recent reference frame in memory to provide a prediction quality that is almost identical to that of MR-FS (about 0.05–0.07 dB loss) and still achieves moderate reductions in computational complexity (a speed-up ratio of about 22–29). Between the two extremes is the MR-SMS algorithm. Compared to MR-FS, it achieves reasonable reductions in computational complexity (a speed-up ratio of about 33–77) with only a slight loss in prediction quality (about 0.1–0.2 dB loss). These observations are further emphasized using Figure 8.11, which compares the performance of the different algorithms when applied to FOREMAN at different frame skips.

A very interesting point to note (from Tables 8.8 and 8.9 and also from Figure 8.11) is that the computational complexity of the multiple-reference SMS algorithms is comparable to (and in some cases less than) that

[9]This excludes the result for AKIYO where $\Delta\text{PSNR} = 0$.



(a) Prediction quality



(b) Computational complexity

Figure 8.11: Comparison between different block-matching algorithms when applied to QSIF FOREMAN with a multiframe memory of $M = 50$ frames



(a) Original frame



(b) Compensated using SR-FS (28.24 dB and 77,439 locations)



(c) Compensated using MR-FS with $M=50$ (31.31 dB and 3,871,950 locations)



(d) Compensated using MR-3DSM with $M=50$ (31.04 dB and 72,532 locations)

Figure 8.12: Subjective quality of the motion-compensated 158th frame of QSIF FOREMAN at 25 frames/s

of single-reference full-search SR-FS, and yet they still maintain the improved prediction gain of multiple-reference motion estimation. This is also illustrated in Figure 8.12, which shows the subjective quality of the motion-compensated 158th frame of FOREMAN. The uncovered background at the bottom-right corner of the frame is poorly compensated using the single-reference algorithm SR-FS (Figure 8.12(b)). This uncovered background is compensated with higher quality using the multiple-reference algorithms (Figures 8.12(c) and 8.12(d)). While the MR-FS algorithm achieves this improved prediction quality at the expense of about 50 times increase in computational complexity, the MR-3DSM algorithm provides a similar improvement at no increase in computational complexity.

8.7 Discussion

There are many techniques for reduced-complexity BMME. The most widely used approach employs a reduced set of motion vector candidates. Algorithms in this category are usually based on a *unimodal error surface assumption*. In most cases, however, this assumption does not hold true, and such algorithms can easily get trapped in local minima, giving a suboptimal prediction quality. The main aim of this chapter was to develop a reduced-complexity BMME that adopts the same approach of reducing the set of motion vector candidates but, at the same time, avoids the local minimum problem.

Thus, the chapter formulated block-matching motion estimation as a two-dimensional constrained minimization problem. It was then proposed to solve this problem, with reduced complexity, using an optimization method called the simplex minimization (SM) optimization method. The resulting solution was called the simplex minimization search (SMS). The initialization procedure, termination criterion, and constraints on the independent variables of the search were designed to take into account the basic properties of the BMME problem.

Simulation results within an isolated test environment showed that the SMS algorithm outperforms other reduced-complexity BMME algorithms, providing better prediction quality, smoother motion field, and higher speed-up ratio. In particular, the SMS algorithm is very resilient to the local minimum problem. This superior performance was also confirmed within an H.263-like codec and an object-based MPEG-4 codec.

It was also noted that the superior performance of the LTM-MCP (discussed in Chapter 6) is achieved at the expense of a significant increase in computational complexity. To reduce complexity, the chapter extended the SMS algorithm to the multiple-reference case. Three different extensions (or algorithms) were presented, each representing a different degree of compromise between prediction quality and computational complexity. Simulation results showed that the multiple-reference SMS algorithms provide significant reductions in computational complexity compared to the multiple-reference full-search. With a multiframe memory of size $M = 50$, the computational complexity of the SMS algorithms is comparable to (and in some cases less than) that of single-reference full-search, and yet they still maintain the improved prediction gain of multiple-reference motion estimation.

Part IV
Error Resilience

When transmitted over a mobile channel, compressed video can suffer severe degradation. Thus, error resilience is one of the main requirements for mobile video communication.

This part contains two chapters. Chapter 9 reviews error-resilience video coding techniques. The chapter considers the types of errors that can affect a video bitstream and examines their impact on decoded video. It then describes a number of error detection and error control techniques. Particular emphasis is given to standard error-resilience techniques included in the recent H.263+, H.263++, and MPEG-4 standards.

Chapter 10 gives examples of the development of error-resilience techniques. The chapter presents two temporal error concealment techniques. The first technique, MFI, is based on motion field interpolation, whereas the second technique, BM-MFI, uses multihypothesis motion compensation (MHMC) to combine MFI with a boundary matching (BM) technique. The techniques are then tested within both an isolated test environment and an H.263 codec. The chapter also investigates the performance of different temporal error concealment techniques when incorporated within a multiple-reference video codec. In particular, the chapter finds a combination of techniques, MFI-BM, that best recovers the spatial-temporal components of a damaged multiple-reference motion vector. In addition, the chapter develops a multihypothesis temporal concealment technique, called MFI-MH, to be used with multiple-reference systems.

Chapter 9

Error-Resilience Video Coding Techniques

9.1 Overview

As already discussed, one of the main requirements for mobile video communication is *error resilience*. When transmitted over a mobile channel, video can be affected by a number of loss mechanisms, like multipath fading, shadowing, and co-channel interference. The effects of such errors are magnified due to the fact that the video bitstream is highly compressed to meet the stringent bandwidth limitations. The higher the compression, the more sensitive the bitstream is to errors, since in this case each bit represents a larger amount of decoded video. The effects of errors are also magnified by the use of predictive and VLC coding, which can lead to temporal and spatial error propagation. It is therefore not difficult to realize that when transmitted over a mobile channel, compressed video can suffer severe degradation, making the use of error-resilience techniques vital. This chapter reviews error-resilience video coding techniques.

The rest of the chapter is organized as follows. Section 9.2 describes the main functional blocks of a typical video communication system. Section 9.3 highlights the main types of errors that can affect a video bitstream. Section 9.4 examines the impact of such errors on the decoded video. Section 9.5 describes a number of error detection techniques. Sections 9.6–9.8 reviews three main categories of error-resilience video coding techniques. The chapter concludes with a discussion in Section 9.9.

9.2 A Typical Video Communication System

Figure 9.1 shows a typical video communication system. The *encoder* consists of a *source encoder* and a *channel encoder*.

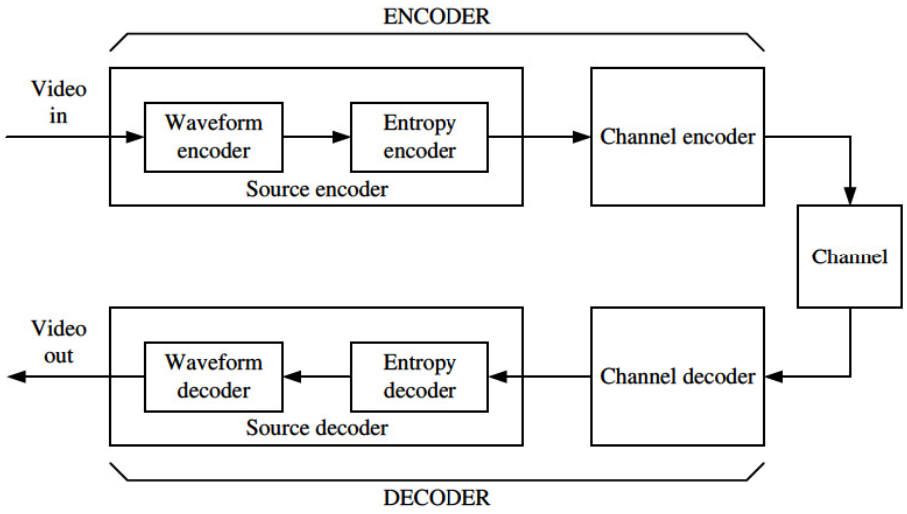


Figure 9.1: Typical video communication system

The function of the source encoder is to compress the input video. It consists of a *waveform encoder* and an *entropy encoder*. The function of the source encoder is described in detail in Chapter 2. With reference to Figure 2.3, the waveform encoder corresponds to the *mapper* and *quantizer* blocks, whereas the entropy encoder corresponds to the *symbol encoder* block. Thus, the waveform encoder works by removing, as much as possible, statistical and psychovisual redundancies present in the input video, whereas the entropy encoder tries to remove coding redundancy.

The channel encoder conditions the compressed bitstream at the output of the source encoder to be suitable for transmission over the channel. This can include, for example, packetization, error protection, modulation, and transport-level control.

At the decoder, the reverse operations are performed to obtain the output video. Note that although this figure shows a one-way communication between the encoder and the decoder, some video communication systems may also have data flowing in the other direction to convey some feedback information.

9.3 Types of Errors

Errors affecting a digital video bitstream can be roughly classified into *two* main categories: *random bit errors* and *erasure errors*.

9.3.1 Random Bit Errors

Random bit errors can occur in the form of bit inversion, bit insertion, and/or bit deletion. They are usually quantified using a parameter called the *bit error rate* (BER), which is the average probability that a bit is in error. Random bit errors are usually caused by physical effects like thermal noise.

9.3.2 Erasure (or Burst) Errors

Erasure errors occur in the form of a loss of (or damage to) contiguous segments of bits. They are usually quantified using parameters like the number of bursts, the length of a burst, and the BER within a burst. Burst errors in a mobile channel can be caused by a number of mechanisms, such as short-term (multipath) fading, long-term (shadowing) fading, and co-channel interference. In a packet-based network, burst errors occur in the form of packet losses due to different reasons, such as congestion, misrouting, and delivery with unacceptably long delays.

It should be pointed out, however, that this classification does not take into account the impact of errors, which is highly dependent on the coding method. For example, it will be shown later that due to the use of predictive and VLC coding, random bit errors in a video bitstream can cause severe error propagation. Thus, random bit errors in a video bitstream are effectively equivalent to burst errors. In what follows, no distinction will be made between the two types of errors, and the generic term *transmission errors* will be used to refer to both types.

9.4 Effects of Errors

Errors occurring in a video bitstream can cause *isolated effects*, *spatial error propagation*, and/or *temporal error propagation*.

9.4.1 Isolated Effects

In this case the effect of an error is limited and does not propagate either spatially or temporally. An example is an error in a FLC codeword. Another example is an error that converts a VLC codeword into another valid codeword of the same length. Note, however, that for both cases to have an isolated effect, it is assumed that the damaged codeword is not a prediction for another codeword and that no temporal error propagation occurs due to motion-compensated prediction. Clearly, such isolated effects are rare occurrences in video bitstreams, and when they do occur their damage is usually acceptable

and can be handled relatively easily. However, such errors can sometimes be catastrophic, as, for example, in the case of errors in vital header information (e.g., frame size, and quantizer step size).

9.4.2 Spatial Error Propagation

This is mainly due to two mechanisms:

1. **Errors in VLC Coded Data**: If an error converts a VLC codeword into an invalid codeword or into a valid codeword of a different length, then this causes *loss of bitstream synchronization*. This can occur in two forms [177]:
   (a) *Loss of Codeword Synchronization*: In this case an error causes the decoder to decode a codeword of the wrong length. As a result, the next codeword will be decoded in the wrong position and all following codewords may be affected. This effect is usually temporary, and the decoder eventually regains codeword synchronization [178].
   (b) *Loss of Coefficient Synchronization*: The second form of loss of synchronization is the loss of coefficient synchronization. Even when codeword synchronization is regained, the decoder will be decoding coefficients that have no meaning without the previous, lost coefficients. For example, in run-length encoding, if an incorrect run-length has been decoded, then all the following data will be misplaced even if it is decoded correctly. Since this form of loss of synchronization usually causes data to be misplaced, it is also referred to as *loss of positional synchronization*.
2. **Errors in Predictively Coded Data**: The second mechanism that causes spatial error propagation is the loss of predictively coded data. For example, a motion vector is usually predictively coded with reference to one or more previous motion vectors. If those previous vectors are in error, then the prediction will be wrong and the errors will propagate to the current motion vector, and so on.

9.4.3 Temporal Error Propagation

This is due mainly to the use of motion compensated prediction (or any other form of predictive coding in the temporal dimension). As already described, in motion-compensated prediction, parts of the current frame are copied (or motion compensated) from a reference frame. If the copied reference parts already contain errors, then those errors will also occur in (i.e., propagate to) the current frame.