Network Driven Motion Estimation for Wireless Video Terminals

by

Wendi Beth Rabiner

B.S., Cornell University (1995)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1997

Author .....
Department of Electrical Engineering and Computer Science
February 1, 1997

Certified by .....
Anantha P. Chandrakasan
Assistant Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chair, Department Committee on Graduate Students

Network Driven Motion Estimation for Wireless Video Terminals

by

Wendi Beth Rabiner

Submitted to the Department of Electrical Engineering and Computer Science
on February 1, 1997, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering

Abstract

Motion estimation has been shown to help significantly in the compression of video sequences. However, since most motion estimation algorithms require a large amount of computation, it is undesirable to use them in power constrained applications, such as battery operated wireless video terminals. This thesis describes a novel motion estimation algorithm, termed network driven motion estimation, which reduces the power dissipation of wireless video devices in a networked environment by exploiting the predictability of object motion. Since the location of an object in the current frame can be predicted from its location in previous frames, it is possible to optimally partition the motion estimation computation between battery operated portable devices and high powered compute servers on the wired network. In network driven motion estimation, a remote high powered resource at the base-station (or on the wired network), predicts the motion vectors of the current frame from the motion vectors of the previous frames. The base-station then sends these predicted motion vectors to a portable video encoder, where motion compensation proceeds as usual. Network driven motion estimation adaptively adjusts the coding algorithm based on the amount of motion in the sequence, using motion prediction to code portions of the video sequence which contain a large amount of motion and conditional replenishment to code portions of the sequence which contain little scene motion. This algorithm achieves a reduction in the number of operations performed at the encoder for motion estimation by over *two orders of magnitude* while introducing minimal degradation to the decoded video compared with full search encoder-based motion estimation.

Thesis Supervisor: Anantha P. Chandrakasan
Title: Assistant Professor

Acknowledgments

My graduate study has been supported through a National Science Foundation Fellowship and a research assistantship from MIT. This work was funded through the ARL Federated Lab program. I would like to thank all the people who were vital in making this thesis come to fruition:

- My advisor, Professor Anantha Chandrakasan, who gave me the freedom to explore my ideas as well as many valuable suggestions for improving our system. Anantha's enthusiasm for the project made this a very fun and gratifying experience. It has truly been a pleasure to work with him.
- Rajeevan Amirtharajah, Duke Xanthopoulos, Jim Goodman, Abram Dancy, and Tom Simon, for answering all my little questions before they became big problems.
- Steve Heinzelman, for always believing in me and making me feel that there is nothing in this world that I cannot accomplish. Steve knows how to make me laugh even when life gets a little crazy, and I thank him for this.
- Most of all, I would like to thank my parents, Suzanne and Larry, for their endless love and support. My father read through numerous articles and abstracts of my work, and I appreciate all the advice and feedback he gave me. My parents have always encouraged and inspired me, and I dedicate this thesis to them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As the use of multimedia devices becomes more prevalent, there is an increasing need to transmit video over low bandwidth networks, such as wireless channels. Due to the large amount of data contained in video signals, efficient compression techniques are extremely important. Conventional video systems use some form of scene motion estimation/motion compensation at the encoder to achieve high compression ratios because it is very effective in decorrelating successive frames. However, most motion estimation algorithms require an extensive search to minimize a distortion metric, which is extremely computationally intensive (e.g., a ± 16 pel full search motion estimation for 30 fps video requires over 65 thousand ops/pixel/second). The high computational requirement of full search motion estimation makes it unsuitable for power constrained portable video terminals, since the power dissipation of the circuit which implements an algorithm is directly proportional to the amount of computation. Minimizing power dissipation is the key to maximizing battery lifetime and thus should be one of the driving forces when designing motion estimation algorithms for portable video devices.

1.1 Low Power Digital System Design

In the past, digital circuit design has been driven by a desire to increase clock speed and reduce chip area at the expense of increasing power. Low power design techniques were used mainly to implement low throughput applications, such as pacemakers, calculators, and wristwatches. However, advances in integrated circuit technology, data compression, and networking have facilitated the wide-spread use of high performance portable devices which integrate data, voice, and images, such as laptop computers and personal communication devices. These devices are typically required to have minimum size and weight while maintaining a fast clock speed. Power reduction techniques are used to minimize the size and weight and extend the battery life of these portable devices.

Power reduction is achieved at all levels of system design, ranging from choosing the technology to implement the circuit to deciding which algorithm to use to accomplish a given function [1, 2, 3]. For example,

algorithms that are highly parallelizeable and have low complexity (i.e., minimum number of computations) are optimal for low power design. In addition, there are architectural and circuit level techniques, such as voltage scaling, data encoding, and gated clocks, which reduce the average power of a digital system. However, minimizing power in one level of system design is not independent from minimizing power in all other levels. An algorithm which requires more computation than another may be optimal if, for example, the more complex algorithm better exploits signal correlations or is highly parallelizeable. This shows that a local increase in power can sometimes be advantageous by achieving a *global* decrease in average power.

Not only does low power design involve minimizing power simultaneously at all levels of the system, it also requires many tradeoffs between performance (circuit speed), chip area, and power. It is important to ensure that power reduction does not substantially reduce performance or increase circuit size beyond what is acceptable. Every stage of the design process requires an optimization of these three parameters, rather than just a minimization of power consumption. This performance-area-power tradeoff is the first of four major design considerations for low power systems described in [2]. The second global consideration is the statistics of the input stream. By exploiting correlations in the data stream or adapting the algorithm to meet the changing statistics, power can be reduced. Avoiding waste is the third challenge of low power system design. This can be achieved by removing unbalanced delays and glitching, powering down unused portions of the circuit, using dedicated hardware, and minimizing the computational complexity of algorithms. The final design goal is to exploit locality, which can reduce power expensive global communications and hence reduce interconnect power.

1.1.1 Techniques for Reducing Power Dissipation

The average power P dissipated by a DSP algorithm is given by

$$P = \sum_i N_i C_i V_{dd}^2 f_s \tag{1.1}$$

where N_i is the number of operations of type i (addition, multiplication, storage, or bus access) performed per sample, C_i is the capacitance switched per operation of type i, V_{dd} is the supply voltage, and f_s is the sample frequency. A main focus of low power design has been the reduction of the supply voltage, as this results in a quadratic reduction in P. Reducing the supply voltage causes increased delays in the circuit, so various architectural designs, such as parallelization and pipelining, must be used to keep a constant throughput as the supply voltage is reduced. A variety of voltage scaling strategies which reduce the supply voltage without any loss in performance (i.e., constant throughput) are described in [1]. The authors in [4] describe an algorithmic approach to maintaining constant throughput using multirate signal processing and parallelization. However, there comes a point beyond which lowering the supply voltage causes too much delay in the circuit to be useful. For example, the techniques required to compensate for the delays introduced by lowering the supply voltage may increase the chip size beyond what is acceptable. Hence, it is also important to reduce the number of operations per sample, N_i, through architectural and algorithmic optimizations.

1.1.2 Computation and Power

At the algorithmic level, the largest reduction in power is obtained by minimizing the total number of operations per sample (N_i). These operations include addition, multiplication, storage, and bus access, all of which consume power. The choice of algorithm is an important example of a high level low power design decision. For example, many low power video coders use vector quantization (VQ) rather than the discrete cosine transform (DCT) used in the MPEG standard since a VQ decoder simply requires a table look-up [5, 6, 7].

Once the algorithm has been chosen, it may be possible to further reduce the number of computations using:

- Mathematical transformations of equations, which involve exploiting mathematical equivalences by rewriting equations in computationally simple forms
- Conversion of multiplications, which require a large amount of power, to low complexity shifts and adds (i.e., computing with powers of two)
- Approximate signal processing algorithms, which trade off a small amount of accuracy for a large gain in power reduction (e.g., filter order reduction)

At a system level, the computation can be optimally partitioned between the portable devices and the high powered network servers to reduce the number of computations in the portable devices. An example of this is the InfoPad terminal [1], which partitions the text/graphics functions such that the portable terminal only performs display updates while a remote high powered server performs the computationally intensive portions of the X-server functions. Network driven motion estimation, the topic of this thesis, uses a similar non-conventional system level approach which exploits network resources to reduce the power consumption of portable video devices.

Table 1.1 highlights the various low power design techniques. By employing these techniques, the power dissipation of digital systems can be minimized.

1.2 Low Power Video Systems

It is very important to design power efficient circuits for portable devices, since this is the key to maximizing battery lifetime. Motion estimation is a vital part of video compression, yet the computational complexity of the full search motion estimation algorithm renders it unsuitable for power constrained video terminals. While work has been done on algorithms which reduce the amount of computation required to perform motion estimation (e.g., hierarchical search, 2D-logarithmic search), the computationally simple algorithms can produce very inaccurate motion estimates whereas the more complex algorithms still require too much power to be suitable for wireless video encoders.

Table 1.1: Low Power Design Techniques.

Design Level	Technique
Logic/Circuit	Clock gating Balance paths to reduce glitching Use static circuit style Minimize device count Exploit signal correlations
Architecture	Concurrency driven voltage scaling Data encoding Dedicated hardware Power-down unused modules
Algorithm	Reduce computational complexity Chose parallelizeable algorithms Mathematical transformations Computing with powers of two Filter order reduction Approximate signal processing
System	**Computation partitioning**

A standard method for reducing the temporal correlation without performing motion estimation is called conditional replenishment. Conditional replenishment consists of representing each frame as the difference between it and the previous frame. This method requires very little computation and does not require motion vectors— the image is represented entirely by the difference image. However, conditional replenishment will only be effective in reducing the temporal correlation if there is little or no motion in the image. If there is a large amount of motion, performing conditional replenishment and sending the difference image may not achieve an acceptable amount of compression of the original image.

Figure 1-1 shows an ultra low power wireless camera. The goals for the design of this system include long battery lifetime as well as high video compression ratios. A novel motion estimation technique, called network driven motion estimation, which transfers the motion estimation computation from the power constrained video terminals to a high powered server on the network has been developed. This motion estimation method produces very accurate motion vectors, which leads to high compression ratios, while minimizing the encoder computation. A software test system which incorporates network driven motion estimation into the H.263 standard has been developed and is used to code various sequences. The results from these experiments show that network driven motion estimation can achieve the power efficiency of conditional replenishment and the bandwidth efficiency of encoder-based motion estimation.

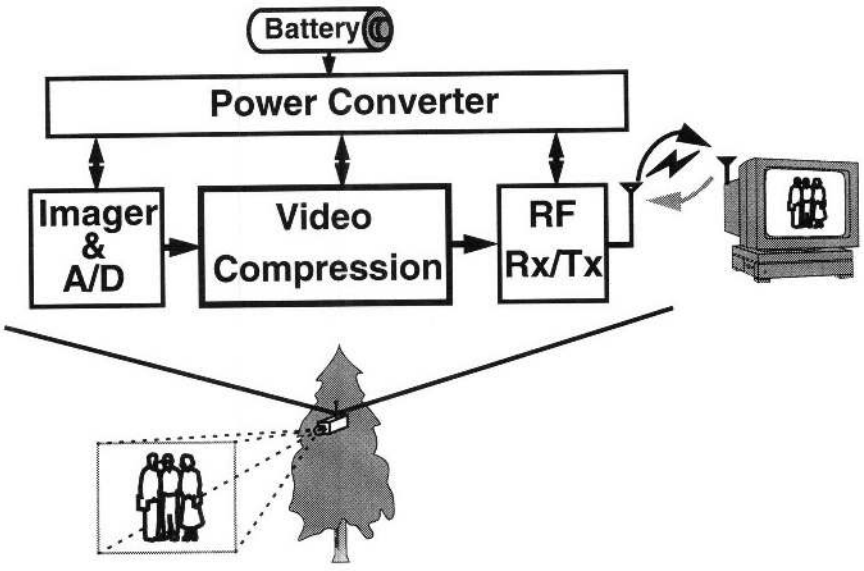


Figure 1-1: Ultra low power wireless camera.

Chapter 2

Motion Estimation

Motion estimation (ME) followed by motion compensation is used to reduce the amount of temporal correlation that exists in most video sequences. This is accomplished by determining, through motion estimation, which portions of an image are related to the previous image. Once the motion of the sequence has been estimated, the previous image can be motion compensated to produce a prediction of the current image. If the motion estimates are relatively accurate, the error between the motion compensated previous image and the original current image will be small. Thus only the description of the scene motion and the small error between the motion compensated previous image and original current image need to be coded, rather than the entire original current image. This achieves a dramatic decrease in the number of bits needed to code each frame. Estimating scene motion can be done on a pixel level, a block level, or a global level. The pel-recursive ME algorithms are very accurate, since the motion is estimated at each pixel in the image, but these algorithms require a large amount of computation and are generally very complex. The block level algorithms may not produce the correct motion estimate for all pixels, but in general the blocks are small enough that the algorithms perform very well. Block-based ME has the added advantage that it is very simple. However, block-based algorithms are computationally intense and do require that overhead motion information be transmitted from the encoder to the decoder. Global ME algorithms require basically no overhead in coding the motion information, since the motion of the entire image is represented by one vector. However, these algorithms try to determine the "average" motion of the sequence which may result in a large residual between the globally motion compensated image and the original image if there are objects moving at different rates within the sequence. Whichever algorithm is used, motion estimation/motion compensation reduces the correlation between successive frames to achieve dramatic compression of the video sequence.

2.1 Pel-Recursive Motion Estimation

Pel-recursive algorithms estimate scene motion by iteratively determining the displacement of each pixel based on previous displacement estimates [8, 9, 10]. The motion estimate is updated for each pixel using the

previous pixel's motion estimate and an update function, which is based on the intensity of the pixels in the current frame which have been transmitted before the current pixel as well as the pixels in the previous frame. Once the pixel motion has been determined, the error between each displaced pixel and the corresponding current pixel is computed to obtain a displaced frame difference (DFD). The update function is assumed to converge quickly to an accurate estimate of the scene motion and hence force the DFD to be zero for a large number of pixels. As the update function only requires the luminance values of causal pixels (i.e., pixels which have been transmitted before, both spatially and temporally, the current pixel), the decoder can compute the update function and estimate the motion for each pixel without the use of motion information transmitted from the encoder. Thus the encoder only transmits the DFD to the decoder.

The pel-recursive algorithms have the advantage of being able to adapt to the motion of different objects within the image, since a new motion estimate is generated for each pixel. Another advantage of pel-recursive algorithms is that they don't require the transmission of motion information, which saves bandwidth over other motion estimation algorithms. However, these algorithms are very complex and require a large amount of computation to update the motion estimate at each pixel. Since no motion information is transmitted from the encoder to the decoder, *both* the encoder and the decoder must perform the motion update computations. While reducing the bandwidth of the coded video, not transmitting motion information doubles the computational intensity of the algorithm. Due to this severe complexity limitation, pel-recursive ME algorithms are not used in power limited video coders.

2.2 Block-Based Motion Estimation

The MPEG standard [11, 12] employs block-based ME [13], as this is the simplest motion estimation algorithm and most easily implemented in hardware. For this technique, each image is broken into 16×16 pixel blocks, called macroblocks. These macroblocks are small enough that there (generally) exists a good match for each macroblock compared with the pixels from the previous frame. This best match is found by comparing each macroblock of the current image to the pixels from the previous image within a certain specified area, called a search window, as shown in Figure 2-1. The size of the search window depends on the sampling rate of the video– typically, for 30 fps video, objects do not move more than ± 16 pels from one frame to the next so the search window $w \times h$ is set to 33×33 pixels[1]. The best match is the set of pixels within the search window from the previous image which is closest, according to some error metric, to the pixels of the macroblock. Common error metrics used in determining the best match are the mean squared error (MSE) and the sum of absolute differences (SAD). If $f(x, y, t)$ represents the luminance value at position (x, y) at time t, the MSE between macroblock i at time t and time $t-1$ is defined as

$$MSE(i) = \frac{1}{16^2} \sum_{y \in MB_i} \sum_{x \in MB_i} [f(x, y, t) - f(x, y, t-1)]^2 \tag{2.1}$$

[1] Occasionally a -15 to +16 pel search is used to make the search window size 32×32.

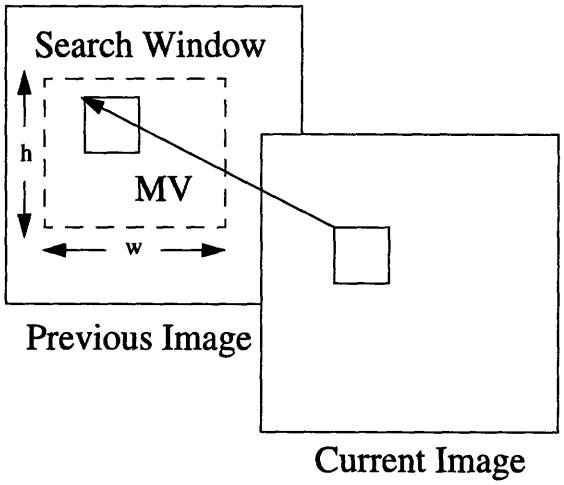


Figure 2-1: Block-based motion estimation.

and the SAD is defined as

$$SAD(i) = \sum_{y \in MB_i} \sum_{x \in MB_i} |f(x,y,t) - f(x,y,t-1)| \tag{2.2}$$

Since the SAD error metric requires no multiplications and does not significantly reduce the accuracy of the motion prediction, it is the preferred error criterion.

A motion vector (MV), which specifies the location of the best match and hence the motion of the macroblock, is generated for each macroblock in the image. In order to guarantee that the minimum error is found, the SAD of every position within the search window needs to be computed. This exhaustive search algorithm, called full search ME, determines the MV for macroblock i as follows

$$\vec{MV}_i = \min_{\{mv_x \in \{-\frac{w}{2}, \frac{w}{2}\}, mv_y \in \{-\frac{h}{2}, \frac{h}{2}\}\}} \sum_{y \in MB_i} \sum_{x \in MB_i} |f(x,y,t) - f(x - mv_x, y - mv_y, t-1)| \tag{2.3}$$

where w is the width and h is the height of the search window. This algorithm guarantees that the MV for each macroblock produces the minimum SAD of all the candidate macroblocks within the search window.

A motion compensated image is created by assembling all the best match macroblocks into an image. This motion compensated image represents a prediction of the current image based on the previous image and the motion of each macroblock. The original image is thus represented by the set of MVs (one for each macroblock) and the motion compensated difference image, which is the residual between the original current image and the motion compensated image. Typically, much less data is required to code the MVs and the motion compensated difference image than is needed to code the original image, so compression is achieved. In conventional video coding systems, such as the encoder shown in Figure 2-2, the encoder performs this motion estimation and computes the motion compensated difference image. The motion compensated difference image is coded using *any* type of video coder (e.g., DCT-based, subband, VQ). Both the MVs and the coded video are sent to the decoder, which uses this information to reconstruct the original image.

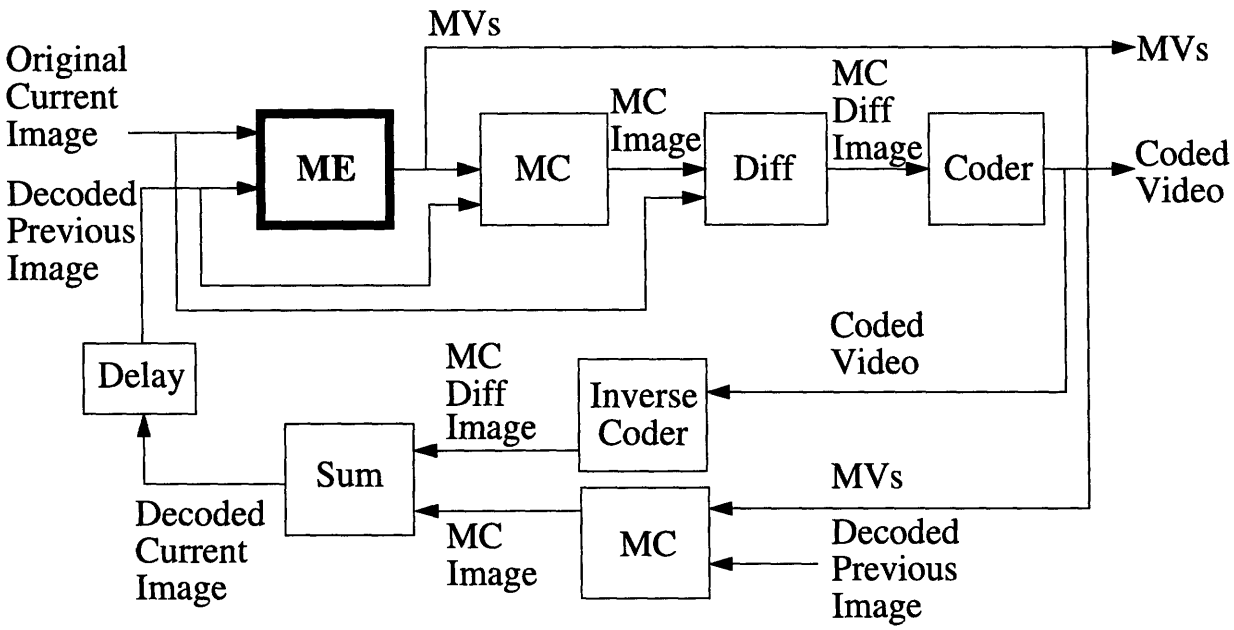


Figure 2-2: Conventional video encoder which uses motion estimation.

2.3 Global Motion Estimation

Global motion estimation (GME) is used to determine a single MV which best represents the dominant scene motion. The previous image is motion compensated using this single vector to produce a globally motion compensated image. Each image is represented using the global MV and the globally motion compensated difference image, the residual between the original current image and the globally motion compensated previous image. The advantage of using GME is that there is very little overhead required to code the motion information, since it consists of only a single vector, and it is very well suited to representing the motion of scenes with panning motion. However, for general scenes the globally motion compensated difference image will typically have much higher energy than the block-based motion compensated difference image. As will be shown in Section 4.1, this increase in energy in the globally motion compensated difference image can be much greater than the savings in bandwidth achieved by representing the motion of the entire sequence by one vector.

Appendix C contains detailed descriptions of two global motion estimation algorithms, the most common MV algorithm and a GME method proposed by Tse and Baker [14]. For both these algorithms, global motion estimation is performed at the encoder using the block-based motion estimates to determine the dominant global motion of the image, and the global motion estimate and the globally motion compensated difference image are sent to the decoder where the original image is reconstructed. Global motion estimation will generally be suboptimal compared with block-based ME since the residual in global motion compensation is typically much larger than the residual in block-based motion compensation. GME also requires more computation than block-based ME, since the block-based estimates must be obtained before the global

parameters can be computed. Since nothing is gained in terms of coding efficiency by using GME alone, it is often combined with local motion compensation to produce a 2-stage motion compensation scheme [14], as shown in Figure 2-3. By combining global and local motion compensation, the global motion effects can first be removed. The local MVs are then computed using the globally motion compensated previous image. Since most of the local MVs will match the dominant scene motion, the size of most local MVs will be reduced, thus improving the coding efficiency of the motion information. This algorithm requires that the block-based MVs be calculated twice, once as the first step in GME, using the previous image, and the second time to determine the local MVs using the globally motion compensated difference image. It is therefore extremely computationally expensive to use this 2-stage global/local motion compensation.

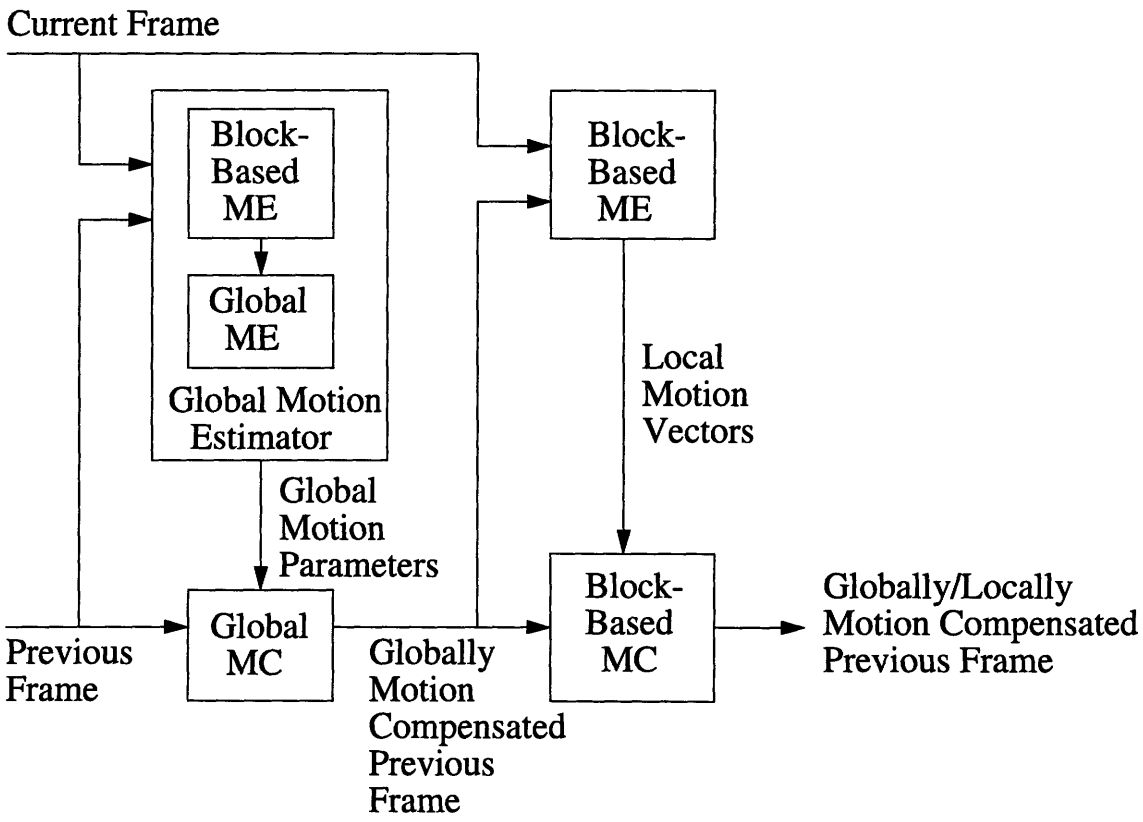


Figure 2-3: Combined global and local motion compensation.

2.4 Computational Complexity of Motion Estimation

Motion estimation followed by motion compensation is very effective in decorrelating successive frames. However, motion estimation is extremely computationally intensive. The pel-recursive techniques require the evaluation of a complex update function, both at the encoder and the decoder, for each pixel. For

full search block-based ME, every macroblock of the image must be compared to every location within the search window. For each comparison, the SAD, which requires 2 addition operations (subtracting the pixel values and adding this difference to the running sum) are performed for all 256 pixels in the macroblock. For a search window of size $w \times h$ (e.g., 33×33 for a ± 16 pel search) and 30 fps video, this requires $60wh$ ops/pixel/second (e.g., 65,340 ops/pixel/second)[2].

Global motion estimation requires even more computation than block-based ME, since the block-based motion estimates must be computed before the global motion estimate is determined. The combined global/local motion estimation algorithm requires more than twice the computation of block-based ME since two sets of local MVs must be computed for each macroblock.

Due to the very high computational requirement of these motion estimation algorithms, they are not suitable for battery operated video systems.

2.5 Reduced Computation Motion Estimation Algorithms

Much work has been done on algorithms which reduce the computational intensity of block-based ME from that of the full search algorithm described in Section 2.2. Many fast algorithms aim to reduce the number of locations at which the SAD is computed. These algorithms operate under the assumption that the SAD decreases monotonically towards the minimum value. However, this assumption is not always true and these algorithms often get trapped in local minima. Thus algorithms which reduce the number of searches for motion estimation, while decreasing the computation, typically produce less accurate MV estimates which increases the bandwidth of the coded video compared with full search ME. Examples of these algorithms include 2D-logarithmic search, three step search, modified conjugate direction search, and hierarchical search [10, 15]. Other fast motion estimation algorithms rely on the correlation that typically exists between MVs of spatially and temporally neighboring macroblocks. In one of the algorithms described in [16], the MV field is reduced to essentially eliminate the search for half the MVs, and the algorithm proposed in [17] estimates each MV from the MVs of the neighboring macroblocks. Finally, algorithms such as the SEA [18] and spiral search use mathematical inequalities to reduce the computation while still guaranteeing that the "true" MV is found.

2.5.1 2D-Logarithmic Search

The 2D-logarithmic search [10] begins by computing the SAD for 5 locations, the center of the search window and the four locations a distance d from the center of the search window, as shown by the •'s in Figure 2-4. If the minimum SAD location is the center point (point A in Figure 2-4) or the minimum SAD location is near the search window boundary, the distance between the search points is decreased to $\frac{d}{2}$ and the SAD is computed for all points a distance of $\frac{d}{2}$ away from this minimum SAD location. Otherwise, the SAD is

[2]This calculation does not take into account the fact that edge macroblocks will have search windows of size less than $w \times h$.

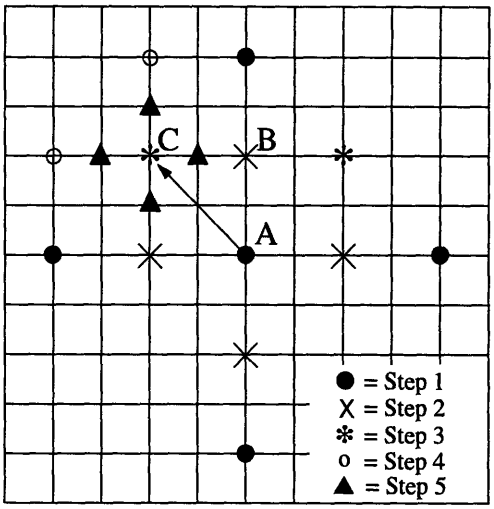


Figure 2-4: Example of a 2D-logarithmic motion vector search.

computed for all the locations a distance d from the minimum SAD location found in the first step. For example, point A in Figure 2-4 is found to have the minimum SAD of the 5 • locations. Since this is the center point, the distance is reduced to $\frac{d}{2}$ for the second search. The ×'s show the location of the search positions for the second search. This search produces point B as the location with minimum SAD, so the third step computes the SAD of the locations specified by the ∗'s, which are all a distance $\frac{d}{2}$ away from point B. The result of this third search is that point C produces the minimum SAD. All locations a distance $\frac{d}{2}$ from C, denoted by the o's, are used for the next search. The result of this fourth search shows that point C still contains the minimum SAD, so the distance is reduced to $\frac{d}{4}$ and the SAD of the points denoted by the △'s is computed. Point C still produces the minimum SAD, and since the distance cannot be reduced any further, point C is the final position, which is locally optimum.

This algorithm requires a varying number of searches, depending on how close the local minimum is to the center of the search window. In general, the 2D-logarithmic search will require at least $4\min(\log_2(\frac{w}{2}), \log_2(\frac{h}{2}))$ (when the minimum SAD value is at the center of the search window) and on average $7\min(\log_2(\frac{w}{2}), \log_2(\frac{h}{2}))$ SAD computations. This is a large reduction in the number of searches compared with full search. However, the tradeoff is that the MV found using the 2D-logarithmic search is often not the "true" MV, which causes the energy in the motion compensated difference image to increase drastically.

2.5.2 Three Step Search

In the three step search algorithm [10], only three steps are taken and the SAD is evaluated at 8 positions for each step (9 positions for the first step). The first step calculates the SAD for the center of the search window and 8 surrounding points, separated from the center by a distance d_1, as shown in Figure 2-5 by the •'s. For this example, the minimum SAD is found at point A. The second step calculates the SAD of 8 positions which are a distance d_2 away from the location of the minimum SAD from step 1, where d_2 is less

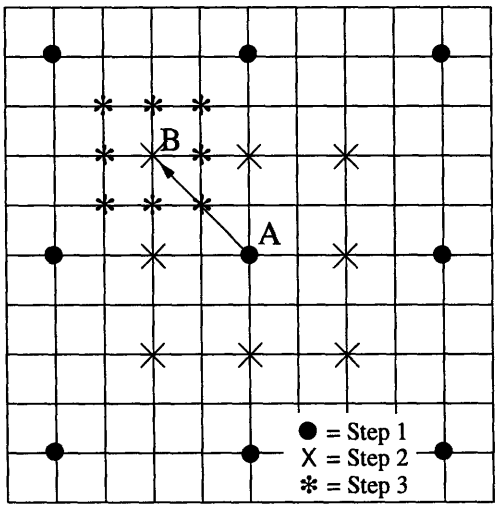


Figure 2-5: Example of a three step motion vector search.

than d_1. In the example, the second step positions are indicated by the ×'s, and the minimum SAD of these points is found to be at point B. The final step calculates the SAD of the locations a distance $d_3 < d_2$ away from the current minimum SAD. The ∗'s in Figure 2-5 represent the search positions evaluated in the third step. Point B is the location of the minimum SAD of all the locations in the third step, so this represents the MV of this macroblock using the three step search.

The three step algorithm requires the evaluation of the SAD at 3 sets of 8 points, plus the center point. This totals 25 SAD computations– a very large reduction in computation compared to full search, but, once again, this algorithm trades off MV accuracy for the reduced computation.

2.5.3 Modified Conjugate Direction Search

The modified conjugate direction search [10] minimizes the SAD separately in the vertical and horizontal directions. The first step determines the minimum SAD in the horizontal direction. This is done by computing the SAD of the three horizontal locations in the center of the search window. The horizontal search continues in whichever direction has the minimum SAD until a point with a SAD less than its two neighbors is found. This is specified as the horizontal component of the MV. The search than determines the SAD of the two vertical neighbors of the minimum horizontal SAD location. The search continues in the vertical direction of minimum SAD until a point is found which has a SAD less than its two vertical neighbors. This point is the vertical direction of the MV. Figure 2-6 shows an example of motion estimation using the modified conjugate direction search. The initial three locations are indicated by the •'s. The left position contains the minimum SAD, so the search continues in the left direction, evaluating the point denoted by the ×; this position has a SAD less than the previous point, so the search continues. The point indicated by the ∗ has a SAD greater than the point denoted by the ×, so the horizontal search is complete, since the SAD of the location denoted by the × is smaller than the SAD of its two horizontal neighbors. The vertical

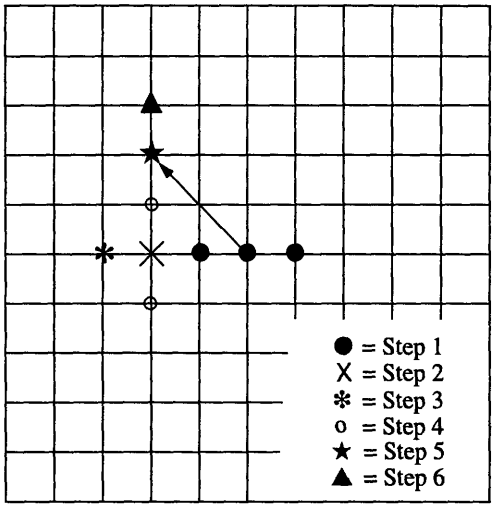


Figure 2-6: Example of a modified conjugate direction vector search.

search continues in the same fashion to find the vertical component of the MV, indicated by the $\star$, the point which has the minimum SAD of its two vertical neighbors.

The conjugate direction search requires at least 5 SAD computations (the center location and the two vertical and two horizontal search locations) and a maximum of $[(\frac{w}{2}+1)(\frac{h}{2}+1)+1]$ SAD computations (if the minimum SAD locations, both horizontally and vertically, are at the search window edges). This is a very small number of computations. However, minimizing the SAD in the horizontal and vertical directions separately can generally give very inaccurate MV estimates.

2.5.4 Hierarchical Motion Estimation

In the hierarchical ME algorithm [15], the image is lowpass filtered and subsampled, possibly several times, to obtain coarser representations of the image. This can be represented by the pyramid structure shown in Figure 2-7. Full search ME is performed using the highest pyramid level (level 2 in Figure 2-7) to obtain a coarse estimate of the MV. This estimate is then refined using the images at all the lower levels of the pyramid. This motion estimation algorithm reduces the computation by requiring a much smaller search window for each of the full searches at the different pyramid levels. The search window is reduced to size $\lceil \frac{w}{2^{2N-n}} \rceil \times \lceil \frac{h}{2^{2N-n}} \rceil$, for each level n of the pyramid, where N is the total number of pyramid levels. For example, if the initial search window is required to be size 33 x 33 and a 2 level pyramid is used (as in Figure 2-7), a full search is performed using the 2^{nd} level image with a search window of size 9 x 9. This MV is then used as the center of the search at level 1, where the search window is reduced to size 5 x 5. Finally the MV is refined using the original image (level 0) and a search window of size 3 x 3.

Hierarchical ME using a 2-level pyramid and a 33 x 33 search window requires 115 SAD computations.

$$\sum_{n=0}^{N} \lceil \frac{w}{2^{2N-n}} \rceil \lceil \frac{h}{2^{2N-n}} \rceil = \lceil \frac{33}{16} \rceil \lceil \frac{33}{16} \rceil + \lceil \frac{33}{8} \rceil \lceil \frac{33}{8} \rceil + \lceil \frac{33}{4} \rceil \lceil \frac{33}{4} \rceil = 115 \tag{2.4}$$

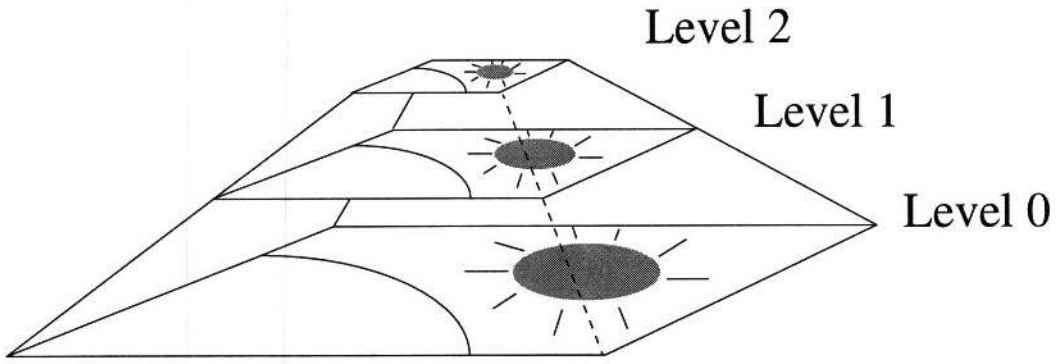


Figure 2-7: Image pyramid for 2 level hierarchical motion estimation.

Hierarchical ME requires much more computation than the methods described above. However, as long as the initial MV estimate, determined in the top pyramid level, is accurate, hierarchical ME will produce much more reliable MVs. The downside of using hierarchical ME is that if the initial MV estimate is inaccurate, the algorithm gets trapped in local minimum and it is almost impossible to find the "true" MV at the lower levels.

2.5.5 Motion Vector Correlation Algorithms

Typically, the pels of moving objects cover more than one macroblock. This implies that MVs of neighboring macroblocks will tend to be correlated. Two algorithms which exploit this MV correlation are described here.

MV field subsampling, as described by Liu and Zaccarin in [16] is an algorithm which only computes half the MVs for each frame. This algorithm finds the MVs for every other macroblock in the image, using a full search algorithm. The MVs for the remaining macroblocks are found by determining which MV of the four surrounding blocks produces the minimum SAD for the middle macroblock. An example of MV field subsampling is shown in Figure 2-8. Motion vectors are computed for each of the shaded macroblocks, and the MV for macroblock A is chosen from the MVs for macroblocks B, C, D, and E, whichever vector produces the minimum SAD for macroblock A. Thus only 4 SAD computations are required to determine half the MVs. Even though the "true" MV is only guaranteed for half the macroblocks, the results in [16] show that this method works extremely well since neighboring macroblocks often have the same MV. The computation for motion estimation using this algorithm reduces to slightly more than 50% of the full search algorithm.

Another MV correlation algorithm, called the interblock correlation method and described in [17], exploits not only the spatial but also the temporal correlation between macroblocks. This algorithm assumes that the four most correlated macroblocks to a macroblock are its immediate temporal neighbor and its spatial neighbors located at an angle of 45°, 90°, and 180°, as shown in Figure 2-9. These macroblocks are all causal neighbors; when the algorithm is determining the MV for a macroblock, the vectors of the four

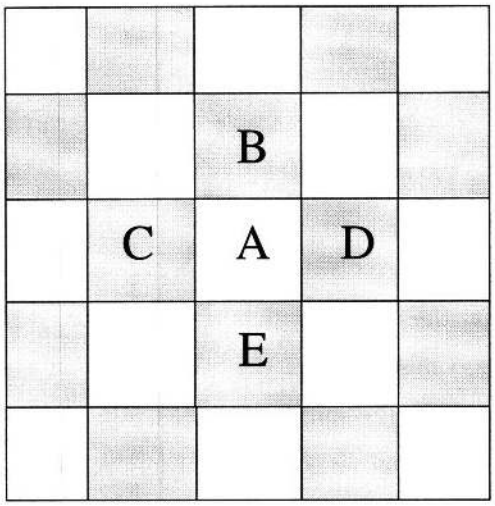


Figure 2-8: Subsampling of the motion vector field.

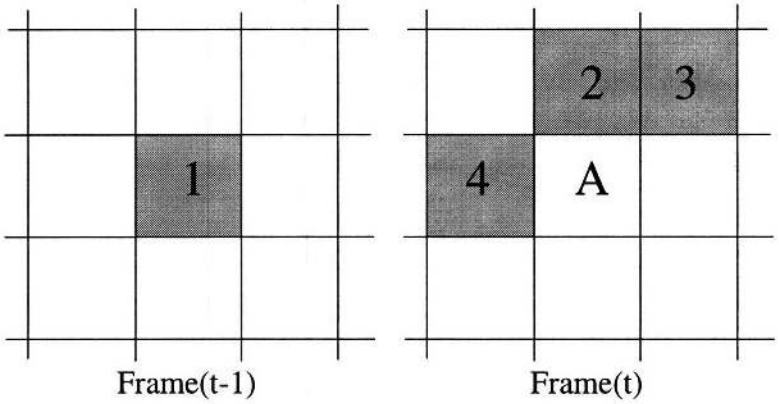


Figure 2-9: Most correlated neighboring macroblocks.

causal neighboring macroblocks have already been found. The algorithm determines which of the four MVs produces a "good" match, defined as a SAD value less than a given threshold. If any of the four candidate MVs satisfy this criterion, it is chosen as the MV for the current macroblock. If not, the best MV of the four (i.e., the one which produces the minimum SAD) is used as the location of the center of the search window and a small search is conducted around this point to find the optimum MV. Computation is greatly reduced using this algorithm since for some macroblocks, only four SAD calculations need to be performed. This algorithm also has the advantage that only 2 bits of MV information need to be transmitted when the chosen MV is one of the four neighboring MVs. As with all the previous algorithms, this algorithm does not guarantee that the "true" MV is found and thus trades MV accuracy for minimization of motion estimation computation.

2.5.6 Mathematical Reduction Algorithms

Another set of fast motion estimation algorithms use mathematical inequalities to reduce the number of computations in the full search algorithm. For example, the spiral search algorithm uses the fact that often the minimum SAD is found near the center of the search window. Thus rather than computing the SAD in a rasterscan order, as shown in Figure 2-10a, the spiral search method begins by computing the SAD at the center of the search window and spiraling outwards, as shown in Figure 2-10b. Once the SAD of a particular location is greater than the current minimum SAD value, that position is no longer a candidate for the optimum location. Thus computation of the SAD ceases once the sum becomes greater than the current minimum. The assumption for the spiral search algorithm is that if the minimum SAD (or a small SAD value) is calculated initially, every other SAD calculation can be stopped early, as most will be greater than this minimum after only a few pixel calculations.

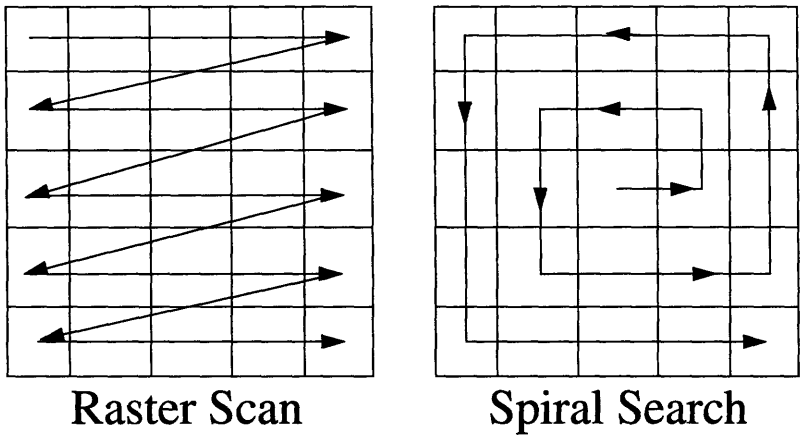


Figure 2-10: a. Raster-scan full search. b. Spiral search algorithm.

The successive elimination algorithm, as proposed by Li and Salari in [18], uses mathematical inequalities to reduce the number of positions within the search window for which the SAD needs to be calculated. In full search ME, the SAD between the current macroblock and each candidate macroblock in the search window is computed. However, the authors in [18] found that certain macroblocks should not even be considered candidates, based on the sum norm of the current macroblock, R, and the sum norm of the candidate macroblock, $S(mv_x, mv_y)$. The sum norm is defined as the sum of the absolute value of all the pixel values in the macroblock.

$$R = \sum_{y \in MB_i} \sum_{x \in MB_i} |f(x, y, t)| \tag{2.5}$$

$$S(mv_x, mv_y) = \sum_{y \in MB_i} \sum_{x \in MB_i} |f(x - mv_x, y - mv_y, t - 1)| \tag{2.6}$$

By exploiting several mathematical inequalities, it was found that only candidate blocks with sum norm

satisfying

$$R - SAD_{min} \leq S(mv_x, mv_y) \leq R + SAD_{min} \tag{2.7}$$

can possibly produce a SAD lower than the current minimum, denoted here by SAD_{min}. Since not all blocks in the search window will satisfy Equation 2.7, this will greatly reduce the number of search positions, particularly if the initial SAD is small. It may appear that this algorithm trades off the calculation of the SAD of each candidate macroblock for the sum norm of the macroblocks. However, the authors in [18] develop a fast algorithm to calculate the sum norm $S(mv_x, mv_y)$ which makes this SEA algorithm much more efficient than the full search algorithm.

2.6 Motion Vector Prediction Algorithms

Motion vectors are highly correlated in time, which implies that the MVs for the current frame can be predicted from the MVs of the previous frames. Not only is this is the idea behind motion vector prediction algorithms, another class of motion estimation algorithms which are used to reduce the encoder computation, but it is also the basis for network driven motion estimation, described in Chapter 3.

Kim and Kuo develop a motion vector prediction algorithm [19] based on the temporal statistics of MVs. This motion estimation algorithm utilizes a search window with a variable size and location, where the search window parameters are determined for each macroblock based on the MVs of the previous frames. Once the search window has been determined, only a few locations within the search window are considered as candidate macroblocks, rather than an exhaustive search of every location within the search window, in order to further reduce the computation.

Assuming that objects move with relatively constant velocity, the current motion can be predicted based on the previous motion of the object. The authors in [19] found that the MV which produces the minimum SAD does not necessarily correspond to the "true" motion of the object. Instead, it is a noise corrupted version of the true MV. Thus rather than just consider the MV which produces the minimum SAD, this algorithm examines the M MVs which produce the M minimum SAD values. Assuming the true object motion is determined (motion vector (mv_x, mv_y)) and the object is moving with constant velocity, the difference in MVs from one frame to the next (for the same macroblock) should be very small.

$$\delta = |mv_x(t) - mv_x(t-1)| + |mv_y(t) - mv_y(t-1)| \approx 0 \tag{2.8}$$

The distribution for δ is assumed to have a peak at 0 and decay as δ becomes larger because the motion vectors are continuous from one frame to the next. Thus the true MVs can be determined by finding the MVs which produce the smallest δ. This is accomplished by finding the M locations with the smallest SAD values using the macroblock at time $(t-1)$ and a search window in the frame at time $(t-2)$ and the M locations with the smallest SAD values using the macroblock at time t and a search window in the frame at time $(t-1)$, as shown in Figure 2-11, where M is equal to 4. These initial MV searches are performed using

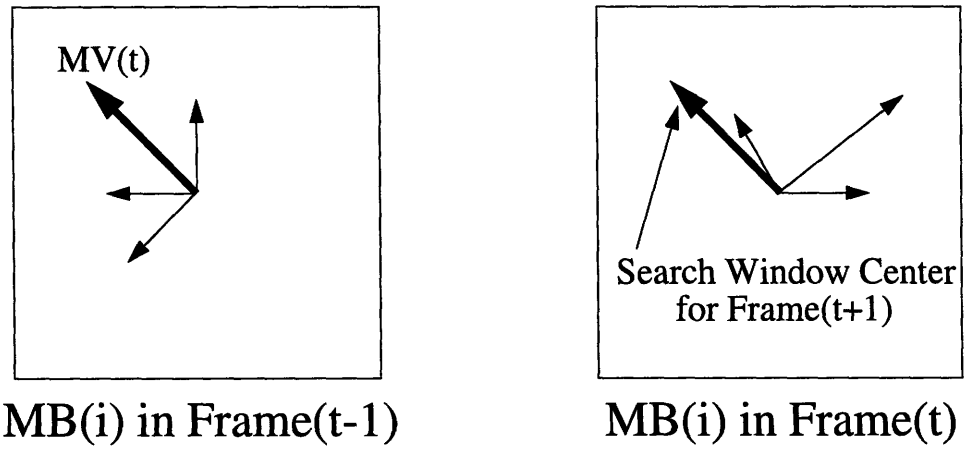


Figure 2-11: Determining the true object motion using the four minimum SAD motion vectors.

exhaustive search ME with a maximum size search window ($w_{max} \times h_{max}$). The distance between all M^2 MVs is computed, and the pair with the smallest distance δ_{min} is selected. One of the MVs of this pair is chosen as the MV for the macroblock at time t and the other is chosen as the center of the search window for the macroblock in the frame at time $(t+1)$. The search window size is set based on the minimum distance δ_{min} according to

$$w = h = w_{min} + 2\delta_{min} \tag{2.9}$$

where w_{min} is the minimum size of the search window. Thus when the minimum distance between the MVs of two consecutive frames is small, the search window for the next frame is greatly reduced, and when it is large, the assumption is that the motion prediction is not very accurate and the search window size is increased to find better MV estimates. A few key positions within the search window are evaluated for the motion estimate of the macroblock at time $(t+1)$; the number of positions evaluated depends on the size of the search window (and hence the size of δ_{min}). The M minimum SAD locations for this search are compared with the M minimum SAD locations from the previous frame and the pair of MVs with the minimum distance of all M^2 locations is chosen. The algorithm continues until δ_{min} is more than 9 pixels, at which point the search window is reset to the original value ($w_{max} \times h_{max}$) and the algorithm starts again with full search ME.

The authors cite an average of 80% to 90% reduction in computation using this MV prediction algorithm (with $M = 4$) rather than full search ME. However, they note that there is a bandwidth tradeoff since the prediction residual is increased.

2.7 Summary

Motion estimation is a necessary step in the coding of video sequences, but it is also the bottleneck in real-time applications, since it requires an excessive amount of computation. Many fast algorithms, such as the ones described above, reduce the amount of computation for motion estimation, but this is usually at the expense of the accuracy of the MV estimates. Some very computationally simple algorithms were described, but these methods can easily get trapped in local minima rather than finding the "true" MV. Algorithms which exploit the spatial and temporal correlation between MVs produce more accurate MVs but require substantially more computation than the simple ME algorithms. The set of fast algorithms which use mathematical relations to reduce computation guarantee that the "true" MV is found, but they do not guarantee that the computation will be reduced very much (or, in the worst case, at all). Table 2.1 summarizes these various fast motion estimation algorithms. This table shows that current motion estimation algorithms which produce accurate MV estimates are very computationally intense. Thus they are unsuitable for use in battery operated wireless video encoders. The following chapter describes network driven motion estimation, an algorithm which rather than reducing the computational intensity of motion estimation, uses remote high powered compute servers on the network to perform this costly task.

Table 2.1: Motion Estimation Algorithms (Search Window $w \times h$, 33×33).

Algorithm	Computation (ops/pixel)		Accuracy
Full Search	$2wh$	1	Optimum
Spiral Search	Variable, $\leq 2wh$	≤ 1	Optimum
Successive Elimination Algorithm	Variable, $\leq 2wh$	≤ 1	Optimum
Predicted MV Algorithm	$0.1(2wh) - 0.2(2wh)$	$0.1 - 0.2$	Medium-High
Interblock Correlation Method	Variable, $\ll 2wh$	$\ll 1$	Medium
MV Field Subsampling	$wh + 4$	0.5	Medium
N-Level Hierarchical ME	$2\sum_{n=0}^{N} \lceil \frac{w}{2^{2N-n}} \rceil \lceil \frac{h}{2^{2N-n}} \rceil$	$(N = 2)$ 0.1	Medium
2D-logarithmic Search	$\geq 8 \min(\log_2(\frac{w}{2}), \log_2(\frac{h}{2}))$ $\leq 14 \min(\log_2(\frac{w}{2}), \log_2(\frac{h}{2}))$	≥ 0.015 ≤ 0.026	Medium
3 Step Search	50	0.023	Low
Modified Conjugate Direction Search	≥ 10 $\leq 2[(\frac{w}{2} + 1) + (\frac{h}{2} + 1) + 1]$	≥ 0.005 ≤ 0.032	Low
Network Driven ME[3]	**18**	**0.008**	**Medium-High**

[2]Network driven motion estimation will be discussed in Chapter 3

Chapter 3

Network Driven Motion Estimation

The previous chapters provided motivation for a low computation video encoder which achieves the high compression rates that conventional video encoders can achieve using full search motion estimation. Many of the fast motion estimation algorithms described in Section 2.5 either produce very inaccurate motion vectors, which drastically increases the bandwidth of the coded video[1], or they do not reduce the computation substantially. This illustrates the tradeoff between coded video bandwidth and encoder computational complexity when performing motion estimation at the encoder. However, if the motion estimation is *removed* from the encoder and performed at a high powered resource on the network, the ill-effects of this tradeoff can be minimized. This is the idea behind network driven motion estimation.

3.1 Networked Wireless Video Systems

A block diagram of a wireless video system in a networked (wired and wireless) environment is shown in Figure 3-1. This system contains a battery-operated video encoder, which acquires, digitizes, and encodes a video signal and transmits the coded video over a wireless link to a base-station. The base-station is connected to a wired backbone and thus is linked to many high powered compute servers by a high bandwidth channel. The base-station sends the coded video to a battery-operated wireless video decoder, which decodes and displays the received images. In conventional wireless video systems, the motion estimation is performed at the encoder, the coded video is sent to the base-station, and the base-station re-transmits the coded video to the decoder. However, since there are often stringent power constraints in the encoder, simple motion estimation algorithms must be used, which reduces the accuracy of the motion vectors and increases the bandwidth of the coded video. Since the wireless channel is a relatively low bandwidth channel, the quality of the decoded images may be highly degraded in order to reduce the bandwidth of the coded video.

[1] Since RF transmit power is proportional to the number of bits transmitted, increasing the bandwidth of the coded video also increases the power dissipation, which may defeat the purpose of using low computation motion estimation algorithms.

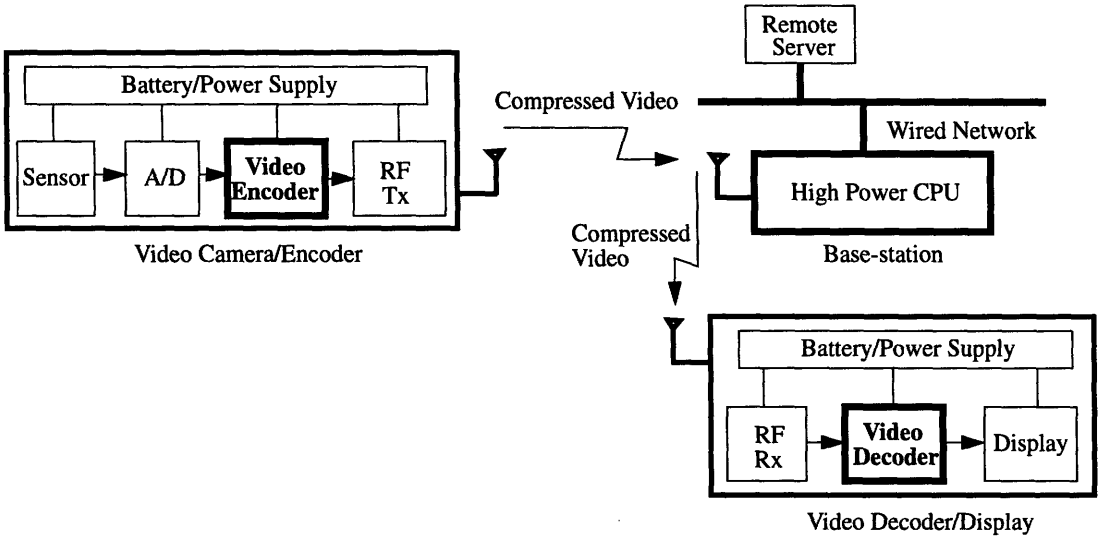


Figure 3-1: Conventional video system in a networked environment.

3.2 Conditional Replenishment

A standard compression method which does not use motion estimation is called conditional replenishment (CR). The idea behind CR is that only pixel values which have changed substantially from the previous frame need to be transmitted. Thus only the difference between the current frame and the previous frame is transmitted. This method requires minimal computation and does not require the use of motion vectors. However, CR will only be effective in reducing the temporal correlation if there is little or no motion in the image. If there is a large amount of motion, performing CR and sending the difference image may not achieve any significant compression. Figure 3-2 shows two consecutive images of the "Bus" sequence and the CR difference image. The CR difference image contains a substantial amount of perceptually relevant information. Thus in order to obtain a high quality reconstructed image, the CR difference image must be coded using a small quantization step size, which will cause the compression ratio to be very low.

The coding can be improved by implementing a small motion estimation search (± 1 pel). This does not require very much added complexity and it can be effective in reducing the bandwidth of the coded video. However, most sequences contain much larger amounts of motion than ± 1 pel between frames. For these scenes, CR or CR with one pel refinement will not be able to achieve acceptable compression ratios.

3.3 Motion Prediction

The motion of objects in natural scenes is mostly continuous from one frame to the next. Therefore, by knowing the location of an object in all previous frames, it is possible to predict its location in the current frame. For example, the location of any object which moves with a constant velocity, such as a moving car, is predictable from two previous frames. Since most objects which do not move with constant velocity

Figure 3-2: Bus Sequence. (a) Previous image. (b) Current image. (c) CR difference image.

accelerate very slowly across the span of a frame in 30 fps video, the constant velocity assumption is a good approximation to the motion of all objects. Motion vectors represent the flow of motion from one frame to the next, and are therefore very correlated in time. This makes it possible to roughly predict the motion vectors of a frame from the motion vectors of the previous frames. The Kim and Kuo motion vector prediction algorithm [19], described in Section 2.6, uses this predictability of the motion vectors to vary the search window size and location in order to reduce the encoder computation for motion estimation. However, the Kim and Kuo algorithm still requires that a substantial amount of computation be performed at the encoder.

A majority of the computation for motion estimation can effectively be *removed* from the encoder by exploiting a high powered remote compute server to perform motion prediction (MP). A resource on the wired network with no power constraints (either at the base-station or a remote location) can estimate the motion of the sequence based on previous decoded frames and use this information to predict the motion vectors of the current frame. A simple yet very accurate method of predicting motion vectors is to have the high powered server find the MVs of the previous reconstructed frame using full search ME (or an equivalent method, such as SEA or spiral search, which guarantees that the MV which produces the minimum SAD is found) and predict that the MVs of the current frame will be the same as the MVs of the previous reconstructed frame. This is similar to the Kim and Kuo MV prediction algorithm, where M is 1 in this case and the motion estimation is performed using the reconstructed images rather than the original images. Kim and Kuo assert that noise will corrupt the determination of the "true" MV and hence they choose the M minimum SAD values as candidate MVs. However, as shown in Figure 3-3, the average bit rate to code various sequences using different values of M in the Kim and Kuo MV prediction algorithm, it is actually slightly detrimental (i.e., increases the bandwidth) to include multiple candidates when searching for the "true" MV. The lowest average bit rate is achieved when $M = 1$, which corresponds to selecting the MV which achieves the minimum SAD as the "true" MV and as the predicted MV for the next frame. Figure 3-4 shows the number of bits required to code each frame of the MPEG "Bus" sequence using a constant quantization step size and both 1 and 4 candidate MVs. This plot shows that the number of bits required to code each frame of the "Bus" sequence is slightly smaller using only 1 candidate MV. However, the difference between using different number of candidate motion vectors (between 1 and 4 motion vectors)

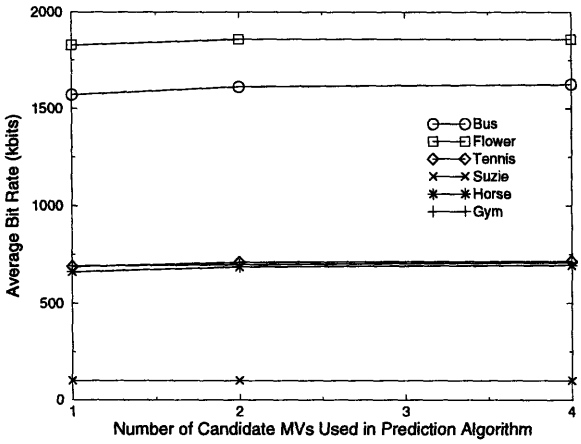


Figure 3-3: Average bit rate using 4, 2, and 1 candidate MVs to predict the current MV.

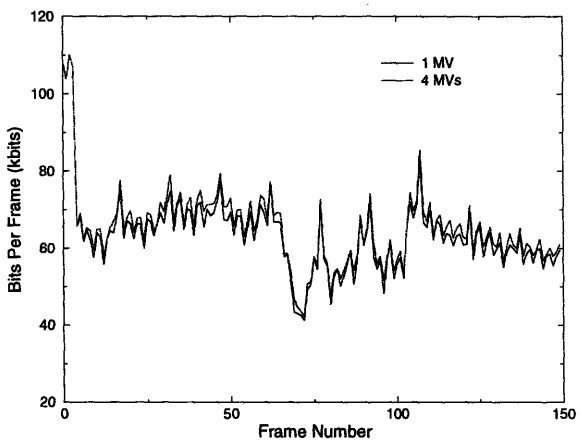


Figure 3-4: Bus Sequence. Bits per frame using 1 and 4 candidate MVs to predict the current MV.

is very small. For the sequences being used in these experiments, the motion vectors which produce the minimum SAD values generally represent the "true" motion (or close to the "true" motion) of the sequences, and including multiple candidate motion vectors actually introduces noise into the motion estimation. Thus it is advantageous to use only 1 candidate MV for the MV prediction algorithm, which is the same as finding the vector which produces the minimum SAD value.

The motion predictor at the high powered network server determines the MVs for each macroblock of the previous reconstructed frame and uses these vectors as the predicted MVs (PMVs) for the current frame. The base-station sends these PMVs to the encoder. However, these vectors are only approximations to the optimal MVs of the current frame. Thus the encoder must perform a local search to improve the MV estimates. The encoder performs motion compensation using these refined PMVs and codes the motion compensated difference image. Both the coded video and the MV refinements are sent back to the base-station, and the current image is reconstructed at the high powered server. The motion predictor can then determine the MVs of the reconstructed current frame and use these vectors as the PMVs for the next frame. Employing motion prediction at a high powered resource rather than motion estimation at the encoder greatly reduces

the computational burden of the encoder. If the motion prediction is relatively accurate, the bandwidth of the coded video will be greatly reduced compared to video coded using encoder motion estimation algorithms of similar computational complexity.

3.4 Adaptive Coding

Motion prediction minimizes the tradeoff between encoder computation and coded video bandwidth by optimally partitioning the motion estimation computation between portable devices and network resources. However, when there is little or no scene motion, it is advantageous to use CR to code the image, rather than predicting the motion (or lack of motion), since CR does not require the transmission of motion vectors. Therefore, it is advantageous to adaptively switch between coding using motion prediction and CR in order to obtain the maximum amount of compression for each frame. This method of reducing temporal correlation using an optimal combination of CR and motion prediction is called *network driven motion estimation.*

The decision about which mode to use for a given frame is made at the base-station (or a remote server) based on a comparison of the variance of the difference image and the variance of the motion compensated difference image of the past frame. This metric is used because the variance of an image is a good measure of the amount of activity in that image [20]. Since the amount of data needed to code each of the difference images is proportional to the amount of activity in these images, the respective variances provide an efficient means of determining which mode should be used. Figure 3-5 shows a plot of the motion compensated difference image variance versus the bit rate required to code the frames of the "Bus" sequence (using a constant quantization step size). This shows that there is a definite correlation between the variance of these images and the number of bits required to code them.

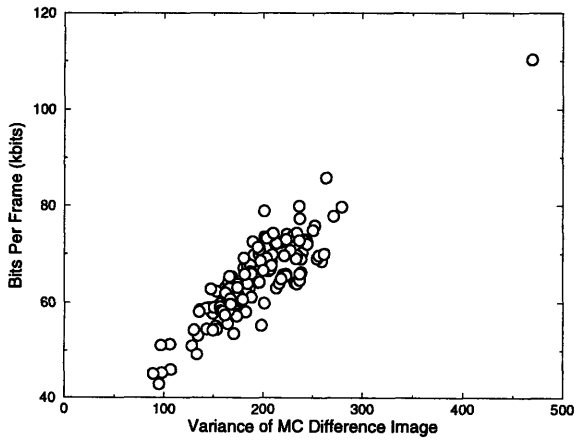


Figure 3-5: Bus Sequence. Bit rate required to code each frame versus the variance of that frame.

3.5 Network Driven Motion Estimation Algorithm

Network driven motion estimation essentially *eliminates* the encoder computational burden for motion estimation, since the encoder only needs to compute a local search to refine the PMVs. Figure 3-6 shows a block diagram of a video coder which uses NDME. This video system is almost identical to the conventional video coder shown in Figure 3-1 except it contains a low bandwidth reverse-channel wireless link and the encoder has both a transmitter and a receiver. Both the motion prediction and the decision of which coding mode to use (CR or MP) are performed at a remote high powered server. The base-station sends the encoder (through the reverse-channel wireless link) either a signal to use the zero motion vector for each macroblock, when the system is operating in the CR mode, or the PMVs, when the system is operating in the MP mode. In either case, the encoder performs a local search centered around the specified motion vectors (either predicted or zero) to find the locally optimum MVs and does motion compensation using these refined MVs. Figure 3-7 shows the average number of bits required to code each frame (using a constant quantization step size) of the standard MPEG sequences "Bus", "Flower", "Tennis", and "Suzie" as well as the sequences "Horse" and "Gym" (obtained using an Accom digital video recorder, as described in Appendix D) versus the number of pels used to refine the PMVs (i.e., k pels translates into a $(2k+1) \times (2k+1)$ pixel search window and $60 \times (2k+1)^2$ ops/pixel/second). This plot shows that there is a large decrease in the average number of bits required to code each frame when the PMVs are refined by ± 1 pel compared with no PMV refinement. However, increasing the search beyond ± 1 pel drastically increases the amount of encoder computation required for the refinement search but does not reduce the data bandwidth significantly. The PMV refinement search is therefore set to ± 1 pel, as this requires only 540 ops/pixel/second and represents the best tradeoff between reducing encoder computation and reducing the data bandwidth.

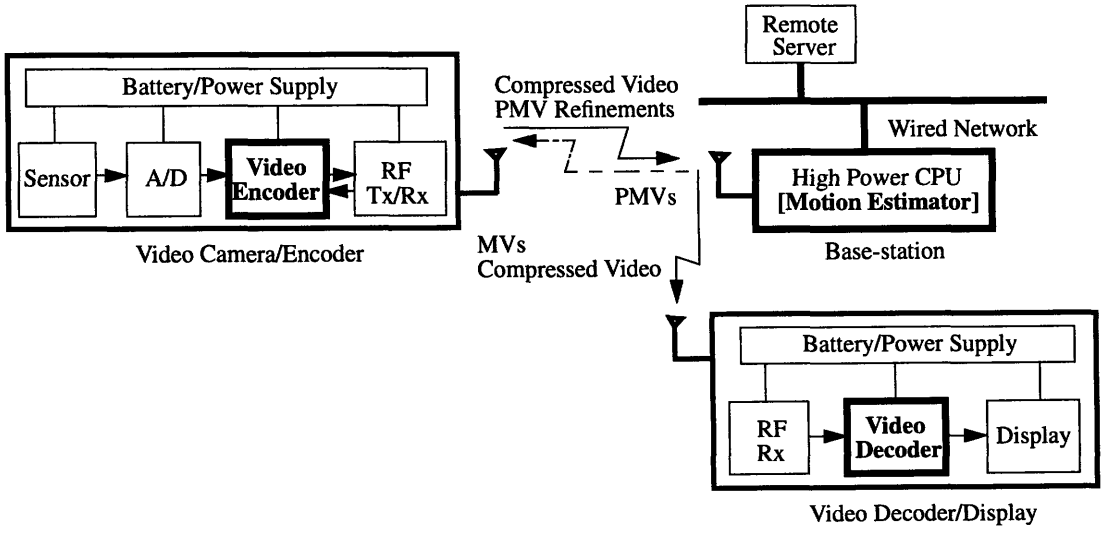


Figure 3-6: Video system which uses NDME.

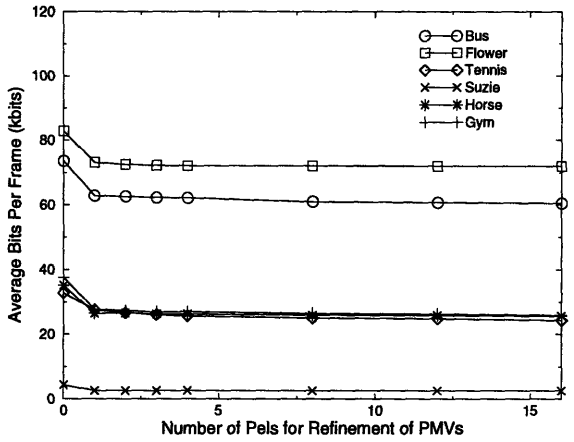


Figure 3-7: Average number of bits per frame versus the number of pixels used in the PMV refinement search.

The encoder transmits the vector refinements and compressed video back to the base-station through the forward channel wireless link, and the base-station transmits the full MVs (the refined PMVs) and the compressed video to the decoder. A high powered server at the base-station (or a remote server) reconstructs the image using the compressed video received from the encoder and performs motion estimation using the current decoded image and the previous decoded image to find the MVs for the reconstructed current frame. Since there are no power constraints, the high powered server uses a spiral search method in order to determine the optimal MVs, which become the predicted MVs for the next frame. The high powered server also determines which mode to use to code the next frame by comparing the variance of the difference image and motion compensated difference image of the reconstructed current frame. The base-station then sends the encoder either a signal to code the next frame using CR or the PMVs for the next frame, and the entire coding process begins again.

3.5.1 Determining Coding Modes

The first frame of a sequence is intra-coded, and the second frame is coded using CR, since the system has no previous frames from which to predict the sequence motion. After these initial frames, the variance metric described above determines which mode is used to code each frame. However, since no PMVs need to be transmitted when the system is operating in the CR mode, this mode is slightly preferenced over MP when there is little scene motion. Similarly, in order to keep the modes relatively consistent from frame to frame, the current mode is preferenced over switching modes. This results in thresholding the difference between the two variances, as shown in Figure 3-8[2]. This method of deciding the system mode continues until a scene change. Since network motion prediction breaks down at a scene change (which is unpredictable), the

[2]T1 was empirically chosen to be 50 and T2 was empirically chosen to be 0. Thus whenever the variance of the difference image is less than the variance of the MC difference image, the system codes the next frame using CR.

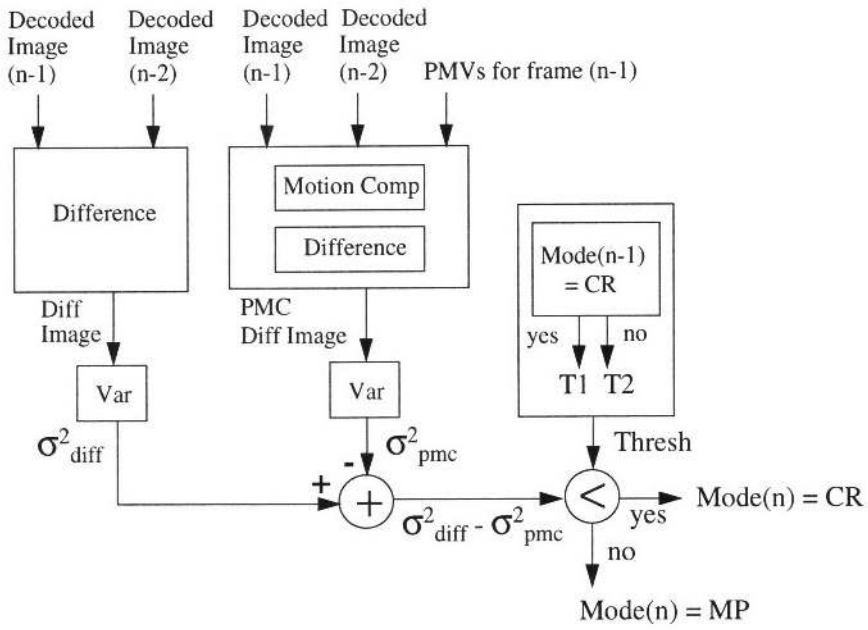


Figure 3-8: Algorithm for determining NDME system mode.

encoder is required to have a simple mechanism for detecting scene changes. For example, whenever the variance of the motion compensated difference image is greater than a threshold, the encoder determines that a scene change has occurred. When a scene change has been detected, the encoder must force an intra coded frame, and the entire network driven motion estimation process begins again, as shown in Figure 3-9.

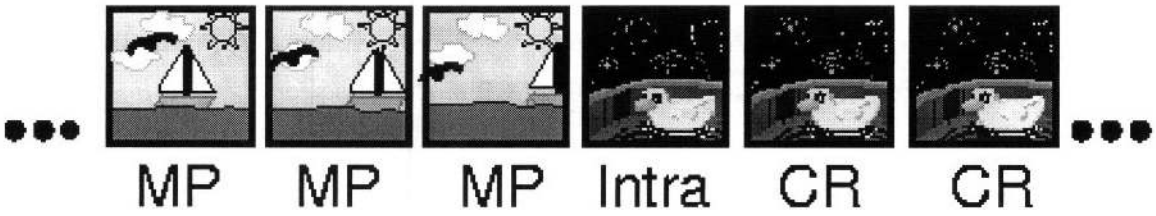


Figure 3-9: Example sequence showing the coding modes for network driven motion estimation. All coding modes are set by a high powered network server except the intra mode, which must be determined at the encoder.

3.5.2 Network Driven Motion Estimation Accuracy

The advantage of using NDME rather than CR can be seen in Figure 3-10, the percentage of motion vectors from the NDME system that are close to the true motion vectors as compared with the percentage of motion vectors from the CR system (1 pel refined (0,0) motion vectors) that are close to the true motion vectors for the "Bus" sequence. There are substantially more motion vectors from the NDME system than from the CR system that are the same as the true motion vectors. Figure 3-10 also shows the percentage of unrefined

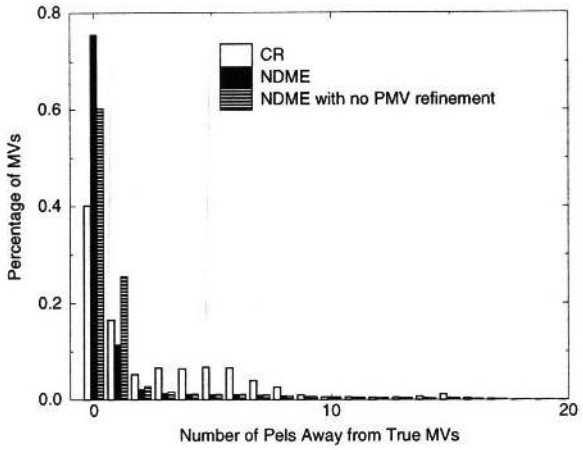


Figure 3-10: Bus Sequence. Distance between 1. CR MVs 2. NDME MVs 3. unrefined PMVs and true MVs.

predicted motion vectors that are close to the true motion vectors. This shows that using unrefined predicted motion vectors produces more accurate vectors than using CR with one pel refinement. However, this also shows the advantage of a ± 1 pel refinement of the predicted motion vectors, as this increases the motion vector accuracy by over 15%[3]. Figure 3-11 also shows the relatively high accuracy of NDME. Figure 3-11a shows the CR with one pel refinement difference image for the two frames of the "Bus" sequence shown in Figure 3-2. There is minimal difference between the CR difference image (shown in Figure 3-2) and the CR with one pel refinement difference image shown here. This is because the motion of the "Bus" sequence is much greater than 1 pixel per frame. However, the motion compensated difference image using NDME, shown in Figure 3-11b for the same two frames, contains substantially less perceptually relevant information than the two CR difference images (Figures 3-11a and 3-2c). The motion compensated difference image using encoder-based full search ME is shown in Figure 3-11c. NDME will achieve almost as much compression of the image as full search ME, since both motion compensated difference images have roughly the same amount of energy. Thus the bandwidth of the NDME coded image will be much less than the bandwidth of the CR coded image but not much more than the bandwidth of the image coded using encoder-based motion estimation.

3.5.3 Video Coders

This NDME system uses the same video encoder as the conventional encoder shown in Figure 2-2 except the NDME system encoder motion estimation is limited to a ± 1 pel search centered around either the PMVs received from the base-station or the zero MV, and the NDME video decoder is the same as a conventional decoder. Thus NDME can be used with *any* type of video coder which is in a networked environment and

[3]Similar histograms for the sequences "Flower", "Tennis", "Suzie", "Horse", and "Gym" are shown in Appendix B. Note that the NDME MVs for sequences such as "Suzie" which do not contain much motion are similar to the CR with 1 pel refinement MVs because many of the frames are coded using the CR mode rather than the MP mode.

Figure 3-11: Bus Sequence. (a) CR with one pel refinement difference image. (b) NDME predicted motion compensated difference image. (c) Encoder based motion estimation motion compensated difference image.

has access to high powered compute servers. The total data bandwidth when the system is operating in the MP mode includes the PMVs being sent from the base-station to the encoder, the corrections to the PMVs being sent back to the base-station, and the compressed video. When the system is operating in the CR mode, the total data bandwidth includes the MV refinements and the compressed video. If the motion prediction algorithm is relatively accurate, the data bandwidth to code each frame using NDME will be close to the bandwidth to code each frame using encoder-based full search ME and much less than the bandwidth to code each frame using CR.

3.6 Network Driven Motion Estimation Constraints

The NDME algorithm requires that predicted motion vectors be sent from the base-station to the encoder. This necessitates a reverse-channel which is capable of transmitting this information. For an image of size $W \times H$, there are $(\frac{W}{16}\frac{H}{16})$ macroblocks in the image, and a MV is generated for each macroblock. For a search window of size $w \times h$, $\log_2 w$ bits are needed to represent the x-component of the PMV and $\log_2 h$ bits are needed to represent the y-component of the PMV. Thus the total number of bits to code the PMVs is

$$bits_{pmvs} = (log_2 w + log_2 h)\frac{W}{16}\frac{H}{16} \tag{3.1}$$

For 30 fps video, the reverse-channel bandwidth (assuming no coding) must be $30bits_{pmvs}$ bits/sec. For CIF images ($W = 352 \times H = 240$) and a search window of size $w = 33 \times h = 33$, the PMVs require a bandwidth of 99 kbits/sec. However, motion vectors of neighboring macroblocks are highly correlated, since often objects are large and cover multiple macroblocks. Thus techniques such as differential coding followed by Huffman coding can substantially reduce the number of bits required to code the PMVs (typically coding can reduce the bit rate to about half of the uncoded rate).

Not only must the reverse-channel have enough bandwidth to send the PMVs, it must also be a very low latency channel. In order for a NDME system to operate in real time, the following steps must be performed in the time span of a single frame:

- High powered compute server on the network determines PMVs and selects coding mode (MP or CR)

- Base-station transmits PMVs or CR signal through reverse-channel to the encoder
- Encoder refines PMVs or (0,0) vector and codes video using refined MVs
- Coded video and vector refinements sent back to the base-station

This requires that both the forward and the reverse wireless channels have very low latency and the high powered compute server has a very high clock frequency. However, as will be shown in Chapter 5, if these constraints can be met, NDME can reduce the encoder power consumption by over *two orders of magnitude* compared with full search motion estimation while retaining the bandwidth efficiency of encoder-based motion estimation.

Chapter 4

Network Driven Motion Estimation Test System

The goal of NDME is to reduce encoder computation as much as possible while keeping the bandwidth of the coded video to a minimum. These are usually conflicting goals, since most coding algorithms require extensive amounts of computation. NDME can achieve encoder computation reduction without a large increase in coded video bandwidth by removing the computationally intense part of the coding procedure, the ± 16 pel search for motion estimation, from the encoder and performing it at a high powered resource on the wired network. It is sometimes desirable to slightly increase the bandwidth of the coded video to achieve a further reduction in encoder power and/or a reduction in the reverse-channel bandwidth. Two methods for accomplishing this were tested: reducing the number of MVs per frame and using the original previous image (rather than the decoded previous image) to perform motion vector refinement and motion compensation at the encoder. These algorithms and the effects they have on the bandwidth of the coded video and encoder computation are discussed in Sections 4.1 and 4.2.

Other algorithms that were developed for the NDME test system include an algorithm to detect scene changes and a rate control module for constant bit rate coding. These algorithms, along with the algorithms to implement NDME, were incorporated into the H.263 video coding standard to create the NDME test system, and a software simulator of this test system was written using the C programming language. A description of the NDME test system, as well as pseudo-code describing these algorithms, is presented in Sections 4.3 - 4.5.

4.1 Image Segmentation

A video system which employs NDME requires a reverse-channel with enough bandwidth to transmit the predicted MVs from the base-station to the encoder. As shown in Section 3.6, this requires a bandwidth of

99 kbits/s for CIF images (assuming no coding). However, if the number of MVs per image was reduced, this bandwidth could be substantially reduced. For example, if only one MV was computed for each frame (i.e., global motion estimation), the total number of bits to represent the PMVs for each frame would be

$$bits_{pmvs} = (log_2 w + log_2 h) \times 1 \tag{4.1}$$

For a search window of size 33 × 33, $bits_{pmvs} = 10$, which, for 30 fps video, is a bit rate of 0.3 kbits/s. This represents a huge savings in the reverse channel bandwidth. However, typically the motion of an image cannot be described very accurately by a single vector, and the bandwidth of the coded motion compensated difference image will be very high. Similarly, the image can be segmented into n regions, as shown in Figure 4-1, where a single motion vector represents the motion in each segment of the image. The total number of bits to represent the PMVs when there are n MVs per image is

$$bits_{pmvs} = (log_2 w + log_2 h) \times n \tag{4.2}$$

The reverse-channel bandwidth therefore increases linearly with the number of MVs per frame.

Two different methods were used to determine the global motion vector for each segment, the most common motion vector (MCMV) algorithm and a global motion estimation algorithm proposed by Tse and Baker in [14]. Both of these algorithms use the local (block-based) MVs to obtain the global motion vector for each segment[1]. Block-based ME is first performed at the decoder to determine the best MV for each macroblock, and the MVs which correspond to the macroblocks in a segment are used to determine the global MV for that segment. It was found that the Tse and Baker algorithm didn't perform substantially better than the MCMV algorithm but did require extensive amounts of extra computation. Thus the experiments were run using the MCMV algorithm to determine the global MV for each segment.

By reducing the number of MVs in each image, the reverse-channel bandwidth is decreased. However, the *total* data bandwidth, which includes the PMVs, the refinements to the PMVs, and the coded video, actually increases by using fewer MVs per frame. Figure 4-2 shows the number of bits required to code each

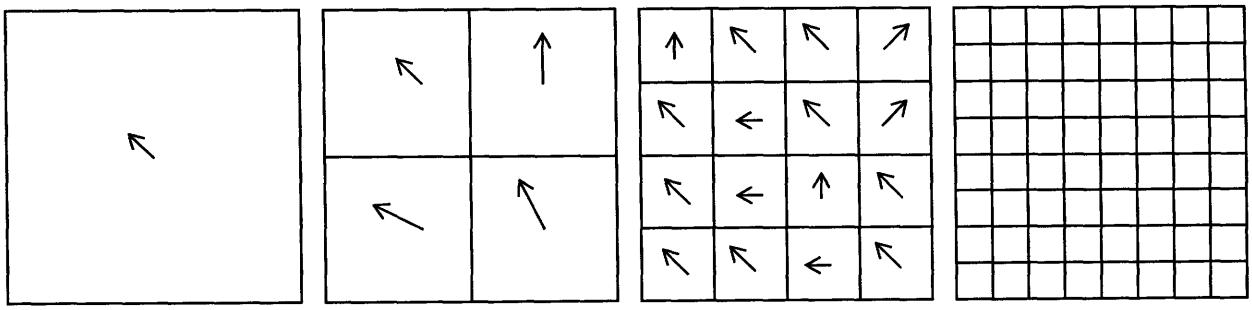


Figure 4-1: Image segmentation to obtain 1, 4, 16, and 64 MVs per image.

[1]See Appendix C for detailed descriptions of these algorithms.

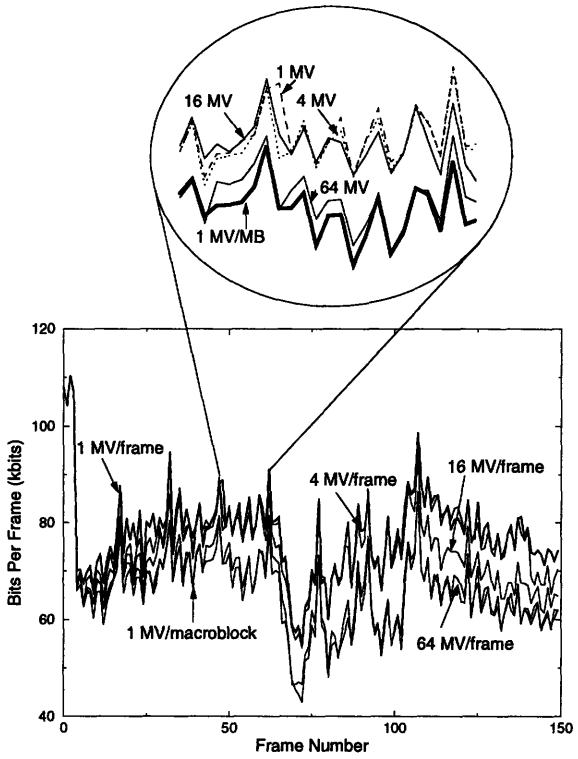


Figure 4-2: Bus Sequence. Bits per frame using different numbers of MVs per frame.

frame of the "Bus" sequence (using a constant quantization step size) for different numbers of segments per frame. The minimum total number of bits is achieved when there is a MV generated for each 16 × 16 pixel macroblock (i.e., 330 MVs per frame, since the images in the "Bus" sequence are size 352 × 240 pixels). The bandwidth for the frames coded using 64 MVs per frame is close to this minimum for this sequence, but the bandwidth for the frames coded using 16 MVs per frame, 4 MVs per frame, and 1 MV per frame are greater than the minimum by as much as 25 kbits/frame. Thus reducing the number of MVs per frame reduces the reverse-channel bandwidth at the expense of a larger increase in the forward-channel bandwidth. Since reducing the number of MVs per frame does not decrease the encoder computation and adds a slight increase in total computation to determine the best MV per segment, there is no advantage in using fewer MVs, unless the reverse channel can only support a small number of bits. The NDME algorithm therefore computes one MV for each 16 × 16 macroblock of the image.

4.2 Motion Estimation/Motion Compensation Using Original Previous Image

One possible method of further reducing encoder computation is to compute the MV refinements and perform motion compensation using the original previous image rather than the reconstructed previous image. By using the original previous image, the encoder does not need to decode each coded image and thus saves

some computation. However, the decoder only has access to the previous reconstructed image. Motion compensating the original previous image will further increase the error of the reconstructed image at the decoder. It may also increase the bandwidth of the coded video because the motion prediction is based on reconstructed images and the predicted MVs will be more accurate using the previous reconstructed frame rather than the previous original frame.

Table 4.1 shows the average bit rate to code the sequences "Bus", "Flower", "Tennis", "Horse", and "Gym" using a constant quantization step size and both the original previous image and the reconstructed image for the MV refinement and motion compensation. This table shows that, as predicted, the average bit rate is lower using the reconstructed previous image rather than the original previous image for MV refinement and motion compensation. However, the differences are very small. Thus it would be acceptable to use the original previous images in the encoder in order to obtain a further reduction in encoder computation when coding the images at these bit rates. However, if the bit rate is reduced and the motion compensated difference image needs to be compressed using a large quantization step size, the reconstructed image may be very different from the original image. In this situation, it is much more desirable to use the reconstructed previous image rather than the original previous image to keep the bandwidth of the coded video to a minimum (since the predicted MVs are determined using reconstructed images) and improve the quality of the reconstructed images. Similarly, errors in the reconstructed image will propagate when motion estimation and motion compensation are performed using the original previous image. Thus the NDME test system uses the reconstructed previous image for MV refinement and motion compensation.

Table 4.1: Average bit rate using a constant quantization step size and either the original previous image or the reconstructed previous image for MV refinement and motion compensation.

Sequence Name	Original Previous Image Bit Rate (kbits/sec)	Reconstructed Previous Image Bit Rate (kbits/sec)
Bus	1590	1570
Flower	1870	1830
Tennis	700	690
Horse	710	660
Gym	710	690

4.3 Detecting Scene Changes

Section 3.5.1 described the need for a simple mechanism at the encoder for detecting scene changes. Frames which occur after a scene change cannot be predicted from frames which come before a scene change, so the encoder must determine that a scene change has occurred and begin the NDME algorithm from an initial intra coded frame. The scene detection algorithm employed in the NDME test system uses the variance of the motion compensated difference image to determine whether or not a scene change has occurred. If this variance is greater than a threshold, the encoder detects a scene change and sends an intra coded frame to the decoder. A good threshold was empirically found to be 500. Using this threshold, the two scene changes in the "Tennis" sequence are detected and no false scene changes are found in any of the other sequences when coding with encoder-based ME and NDME. However, when coding the sequences using CR, the variance of the difference image is often over 500, even if there is no scene change, because CR does not model the scene motion and can produce difference images with very high variances. This causes many false scene detections in CR coded video. To avoid this, the scene change detection threshold is set to 1000 for images coded using CR. However, with the threshold set this high, the scene changes in the "Tennis" sequence are not detected. Figure 4-3 shows the variances of the predicted motion compensated difference images and the CR difference images for the "Tennis" sequence. This shows that there is a huge increase in the variance of the predicted motion compensated difference image when the scene changes occur (frame 89 and frame 148) yet there is almost no perceptible change in the CR difference image when the scene change occurs. This is because NDME attempts to determine the scene motion. Once a scene change occurs, the motion prediction is not valid and the variance of the motion compensated difference image using the predicted motion vectors is very high. CR, on the other hand, does not give a good prediction of the current image when there is any substantial scene motion and can produce difference images which have very high variances regardless of whether or not there is a scene change. Thus another advantage of using NDME over CR is that scene changes are much easier to detect using NDME.

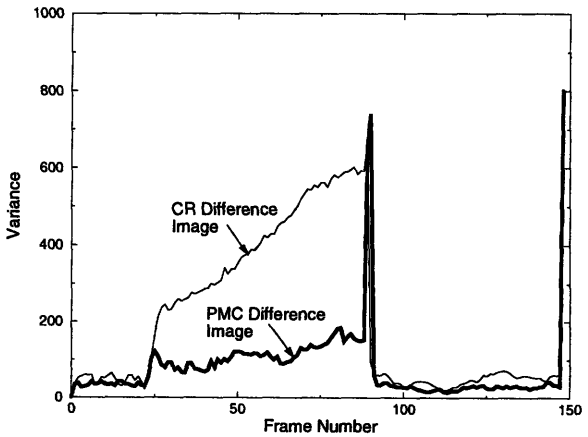


Figure 4-3: Tennis Sequence. Variance of CR and PMC difference images.

4.4 Constant Bit Rate Coding

In order to see the effects of these different coding algorithms on image quality, the sequences were coded using a constant bit rate and a varying quantization step size. The H.263 coding standard specifies a rate control module which sets the quantization step size at each macroblock row to meet the desired bit rate. However, if an image uses more than the allocated number of bits, there will be too few bits for the next image and the rate control module decides that the next few frames must be skipped in order to keep the overall bitrate of the sequence constant. This means that the rate control uses both spatial and temporal quantization in order to obtain a constant bit rate stream. Because the H.263 rate control can drop an unspecified number of frames at any point, it is not suitable for a system which uses NDME, since motion prediction relies on the fact that there is a constant sampling rate between the frames. The predicted motion vectors are intended for the next frame and will not be accurate if they are used two frames into the future rather than for the next frame. Thus the NDME system needs a rate control which maintains a constant bit rate and a constant frame rate.

The rate control used in the NDME test system determines the ideal number of bits to use to code each frame. For example, if the bit rate B is set to 1 Mbps and there are 30 fps, the desired bit rate per frame, B_f, is

$$B_f = \frac{B}{f_s} = \frac{10^6}{30} = 33,330 \;\; b/frame \tag{4.3}$$

The quantizer is set so that approximately B_f bits will be used to code the current frame[2]. However, this will not be exact and the coder will actually require B_1 bits to code frame 1. This leaves a total of $B - B_1$ bits to code the remainder of the frames in that group (i.e., the remaining 29 frames). Thus the desired number of bits per frame for the remaining frames is

$$B_f = \frac{B - B_1}{f_s - 1} \tag{4.4}$$

In general, the desired number of bits per frame for the n^{th} frame ($n \leq f_s$) is given by

$$B_f = \frac{B - \sum_{i=1}^{n-1} B_i}{f_s - (n-1)} \tag{4.5}$$

If too many bits are used to code the $(n-1)^{st}$ frame, the quantizer step size is increased and the n^{th} frame is quantized more coarsely to ensure that fewer bits are needed to code the image. If any of the frames use so many bits that the number of bits remaining for the rest of the group of images becomes negative, the quantization step size is set to 31 (the maximum value) for the remainder of the images in the group. Once the last frame in the group (i.e., frame 30) has been coded, the number of bits remaining increases by B. If there are still bits remaining from the last group, these get added to B and the next group of 30 frames can use more bits. If, on the other hand, the last group used more than the allotted B bits, the excess is subtracted from B and the next group of 30 frames must use fewer bits.

Chapter 5 and Appendix B shows images coded using this rate control module.

[2]See the MPEG standard [11] for information about setting the quantization step size given the desired number of bits for the frame.

4.5 Test System Code

Network driven motion estimation was incorporated into a C software video coder which produces H.263 bit streams. This coder was written by Karl Olav Lillevold from Telenor R&D. Table 4.2 highlights the input options for running this video coder. The H.263 coder is DCT-based, which means that each image to be coded (i.e., the motion compensated difference image or conditional replenishment difference image) is transformed using a 2 dimensional discrete cosine transform (DCT) and the DCT coefficients are then quantized and coded using a Huffman coder. The quantization is either constant for every coefficient, when the coder is operating in the constant quantization step size mode, or it is variable and set according to the rate control module described in Section 4.4. The first image is always intra coded. This means that the original image is transformed using the DCT and the DCT coefficients are quantized and encoded. The following frames are all coded using some form of motion estimation, which is chosen by the user (option -c). The coder can use encoder-based ME (-c 0), which uses a full search method to determine the MVs; predicted ME (-c 1), which predicts the MVs for every frame; CR (-c 2), which uses 1 pel refinement of the (0,0) MV for each macroblock; and NDME (-c 3), which adaptively switches between coding the frames using motion prediction and CR. For each of the motion estimation methods, MVs are found for each 16×16 pixel macroblock and the image is motion compensated using these motion vectors. The motion compensated difference image is then transformed using the DCT and the DCT coefficients are encoded. Typically much

Table 4.2: Parameters for the H.263 (with NDME extension) video coder.

Option	Description	Default Value
-a <n>	start frame number	0
-b <n>	end frame number	0
-c <n> (0...3)	ME mode: 0=EBME, 1=PME, 2=CR, 3=NDME	3
-s <n> (0...16)	integer pel search window size	16
-q <n> (0...31)	quantization step size (QP)	10
-r <n>	bitrate in bits/s (constant bitrate)	variable bitrate (constant QP)
-f <n>	frame rate in frames/s (constant bitrate coding)	30 Hz
-i <filename>	original sequence name	~wendi/elwood/images/bus/bus
-o <filename>	reconstructed frames	./out.recon
-B <filename>	filename for bit stream	./stream.263
-w	write out difference image file to stream ./diff	N/A
-h	print help message	N/A

more data is required for the first intra coded frame than for the remaining frames in the sequence because the different motion estimation/motion compensation methods remove the temporal correlation between frames and achieve substantial amounts of compression. A detailed description of the H.263 coding standard and this particular H.263 software coder can be found on-line at

```
http://www.fou.telenor.no/brukere/DVC/h263_wht/
```

Several functions were added to this encoder to provide NDME capability as well as to implement the scene change detector, rate control module, and various coding functionalities.

4.5.1 Find True/Predicted Motion Vectors

The high-powered server is required to find the optimum MVs for each reconstructed frame and transmit them to the encoder as the predicted MVs for the next frame. These MVs are found using the full search algorithm illustrated here in pseudo-code.

```
cur = read_reconstructed_image(frame_no);
prev = read_reconstructed_image(frame_no-1);
search = 16;
for (mby = 0; mby < number_MB_rows; mby++)
  for (mbx = 0; mbx < number_MB_cols; mbx++) {
    x = MB_SIZE*mbx;
    y = MB_SIZE*mby;
    macroblock = GetBlock(cur, x, y, x+MB_SIZE, y+MB_SIZE);
    search_area = GetBlock(prev, x-search, y-search, x+search+MB_SIZE, y+search+MB_SIZE);
    min_error = INT_MAX;
    for (j = -search; j <= search; j++)
      for (i = -search; i <= search; i++) {
        error = ComputeSAD(macroblock,search_area(i,j))
        if (error < min_error) {
          min_error = error;
          pos_x = i;
          pos_y = j;
        }
      }
    MV_x(mbx,mby,frame_no) = pos_x;
    MV_y(mbx,mby,frame_no) = pos_y;
  }
```

This algorithm computes the SAD value of every candidate macroblock in the search window, which requires over 65,000 operations/pixel/second. However, for NDME this computation is being performed at a high-powered server on the wired network, where computational complexity is not a constraining factor. Thus the MV which produces the minimum error of all possible vectors is chosen as the MV for each macroblock.

4.5.2 Determine Coding Mode

Once the MVs for the current reconstructed frame have been found, they become the predicted MVs for the next frame. The base-station either sends these predicted MVs to the encoder or sends the encoder a signal to use CR, based on the variance of the motion compensated difference images. The following pseudo-code shows how the high powered server determines the coding mode.

```
cur = read_reconstructed_image(frame_no);
prev = read_reconstructed_image(frame_no-1);
diff = abs(cur - prev);
dvar = ComputeVar(diff);

mcimage = FindMCImage(prev, PMV(frame_no));
mcdiff = abs(cur - mcimage);
mcvar = ComputeVar(mcdiff);

if (prev_mode == CR)
  if (mcvar  < dvar - PREF_CR_MODE - PREF_ACTIVE_MODE)
    coding_mode = MP;
  else
    coding_mode = CR;
else
  /* prev_mode is motion prediction (MP) */
  if (mcvar - PREF_ACTIVE_MODE < dvar - PREF_CR_MODE)
    coding_mode = MP;
  else
    coding_mode = CR;

if (coding_mode == MP) {
  /* set the predicted motion vectors for the next frame */
  for (mby = 0; mby < number_MB_rows; mby++)
    for (mbx = 0; mbx < number_MB_cols; mbx++) {
      PMV_x(mbx,mby,frame_no+1) = MV_x(mbx,mby,frame_no);
```

```
      PMV_y(mbx,mby,frame_no+1) = MV_y(mbx,mby,frame_no);
    }
  SendEncoderPMVs(PMV(frame_no+1));
} else
  SendEncoderCRSignal();
```

The coding mode is determined based on the variance of the motion compensated difference image and the variance of the conditional replenishment difference image. If there is motion in the image, the motion compensated difference image variance will be much lower than the conditional replenishment difference image variance. The assumption is that the next frame will exhibit similar motion to the current frame and thus the motion of the current frame can determine which mode should be used to code the next frame. However, since MVs do not need to be sent to the encoder when the mode is set to CR, this mode is slightly preferenced over the MP mode in order to save bandwidth in the reverse channel. Similarly, in order to keep the coding modes consistent from frame to frame, the previous mode is preferenced over switching modes. Good values for these parameters were empirically determined to be PREF_ACTIVE_MODE = 25 and PREF_CR_MODE = 25. Thus the variance of the motion compensated difference image must be at least 50 larger than the variance of the difference image in order to change the coding mode from CR to MP. However, the coding mode will be changed from MP to CR once the variance of the difference image is less than the variance of the motion compensated difference image. Once the mode has been determined, the correct information is transmitted to the encoder.

If the coding mode is set to MP, the predicted MVs must be sent to the encoder. In order to minimize the bandwidth of this transmission, these vectors are coded using a differential coding method followed by a Huffman code. The differential method is based on the fact that spatially neighboring macroblocks typically have similar motion vectors, as discussed in Section 2.5.5. Since the Huffman coder assigns shorter codewords to smaller vector lengths, ideally the encoded values should be as small as possible. Neighboring macroblocks have similar MVs, so this minimization can be achieved by sending the difference between the MV of the current macroblock and the MV of a neighboring macroblock. The method used in this coder chooses the closest MV from one of the causal neighbors shown in Figure 4-4, since these are the most spatially correlated causal macroblocks to the center macroblock. The chosen MV for the differential coding of the current macroblock is the one which has the average value of the three candidate MVs. For example, the chosen MV for the differential coding of macroblock A is computed according to

$$\begin{aligned} C_x(A) &= MV_x(1) + MV_x(2) + MV_x(3) \\ &\quad - \max(MV_x(1), MV_x(2), MV_x(3)) \\ &\quad - \min(MV_x(1), MV_x(2), MV_x(3)) \end{aligned} \tag{4.6}$$

$$\begin{aligned} C_y(A) &= MV_y(1) + MV_y(2) + MV_y(3) \\ &\quad - \max(MV_y(1), MV_y(2), MV_y(3)) \\ &\quad - \min(MV_y(1), MV_y(2), MV_y(3)) \end{aligned} \tag{4.7}$$

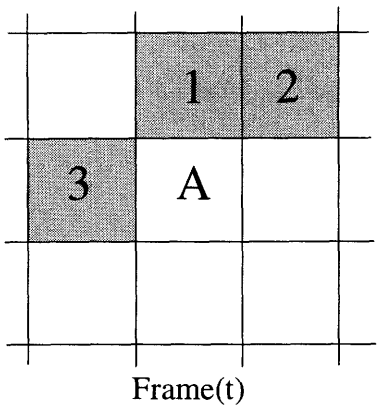


Figure 4-4: Most correlated neighboring macroblocks.

Since the MVs for macroblocks 1, 2, and 3 are known while the MV for macroblock A is being decoded, no further information needs to be transmitted to determine $\vec{C}(A)$. The MV with the median value is chosen as $\vec{C}(A)$ because it is most likely to be the closest of the three MVs to the MV of A. Thus, rather than coding the value for the MV of A, the difference between $\vec{C}(A)$ and $\vec{MV}(A)$ is coded. This value should be small for most motion vectors and thus a short codeword will be used to describe this difference. Table A.1 shows the Huffman code used to represent the difference between $\vec{MV}(A)$ and $\vec{C}(A)$. Since MVs can have values from -16 to +16 ($\pm$16 pel search window), the difference between two motion vectors can have a value from -32 to +32. The Huffman coder assigns the appropriate codeword to this difference using Table A.1, and these coded differentials are sent as the compressed predicted MVs from the base-station to the encoder[3].

4.5.3 Refine Motion Vectors

Once the encoder receives the predicted motion vectors (or the signal to use CR to code the image), it performs a local search to refine the motion estimates. When the MP mode is used to code the image, this refinement is centered around the predicted MVs, and when the CR mode is used, the refinement is centered around the zero MV. In either case, a $\pm$1 pel search is conducted to find the local optimum MV.

```
if(coding_mode == CR)
  MV = Zero;
else
  MV = PMV(frame_no+1);
cur = read_original_image(frame_no+1);
prev = read_reconstructed_image(frame_no);
sxy = 1;
```

[3]The smaller values are represented with shorter codewords because they have the highest probability of occurring due to the differential coding and the correlation of neighboring motion vectors.

```
  for (mby = 0; mby < number_MB_rows; mby++)
    for (mbx = 0; mbx < number_MB_cols; mbx++) {
      x = MB_SIZE*mbx;
      y = MB_SIZE*mby;
      mvx = MV_x(mbx,mby);
      mvy = MV_y(mbx,mby);
      macroblock = GetBlock(cur, x, y, x+MB_SIZE, y+MB_SIZE);
      search_area = GetBlock(prev, x-mvx-sxy, y-mvy-sxy, x-mvx+sxy+MB_SIZE, y-mvy+sxy+MB_SIZE);
      min_error = INT_MAX;

      for (j = -search; j < search; j++)
        for (i = -search; i < search; i++) {
          error = ComputeSAD(macroblock,search_area(i,j))
          if (error < min_error) {
            min_error = error;
            pos_x = i;
            pos_y = j;
          }
        }

      refine_x(mbx,mby) = pos_x;
      refine_y(mbx,mby) = pos_y;
MV_x(mbx,mby) += pos_x;
MV_y(mbx,mby) += pos_y;
}
```

The motion vectors MV(mbx,mby) are used to perform motion compensation. However, only the vector refinements need to be transmitted to the base-station since the base-station already knows the predicted MVs. The refinements will either be 0, 1, or -1, so they are directly Huffman coded rather than using a differential coder before the Huffman coder, since differential coding would increase rather than compress the values (i.e., differential coding produces numbers from -2 through 2, all of which are equally likely, whereas the refinement values are only from -1 through 1). The Huffman table used to code the MV refinements is shown in Table A.2. The maximum number of bits required to code each refinement vector is 4 (2 bits for the x-component and 2 bits for the y-component). However, many of the refinements will be zero (which only requires 1 bit to code) and thus some compression will be achieved.

4.5.4 Check for Scene Change

Before the encoder transforms and codes the motion compensated difference image (using the refined MVs), it checks to determine if a scene change has occurred. This is done by computing the variance of the motion compensated difference image and comparing it to a pre-defined threshold. If the variance is above this threshold, a scene change has occurred and the encoder transforms and quantizes the original image rather than the motion compensated difference image. The entire coding process begins again when a scene change occurs (i.e., an intra-coded frame followed by a CR coded frame followed by either MP coding or CR coding, determined by the base-station). The following pseudo-code shows the method for detecting scene changes.

```
cur = read_original_image(frame_no+1);
prev = read_reconstructed_image(frame_no);

mcimage = FindMCImage(prev, MV);
mcdiff = abs(cur - mcimage);
mcvar = ComputeVar(mcdiff);

if (mcvar > scene_change_thresh) {
  scene_change = 1;
  coding_mode = INTRA;
}
```

If a scene change is detected, the image is intra coded. Otherwise, the motion compensated difference image is transformed and quantized, using the H.263 encoder, and the quantized DCT coefficients are encoded using a Huffman coder as specified by the H.263 standard. The coded video and the MV refinements are sent to the base-station, where the image is reconstructed and the algorithm begins again by determining the MVs of this new reconstructed frame and sending them as the predicted MVs of the next frame.

Chapter 5

Experimental Results

To test the effectiveness of NDME, several experiments were run using 150 frames of the standard sequences "Bus", "Flower", "Tennis" (all 352 × 240 pels), and "Suzie" (176 × 144 pels), as well as 150 frames of the sequences "Horse" and "Gym" (both 352 × 240 pels) obtained using an Accom digital video recorder. In the uncoded images, there are 8 bits/pixel to represent the luminance (255 luminance values) and 2 bits/pixel to represent each component of the chrominance, since the chrominance is subsampled vertically and horizontally. Thus for the 352 × 240 pel images, the total bit rate for the uncoded video is

$$B_{uc} = (352 \times 240)\ pixels \times 12\ bits/pixel \times 30\ fps = 30\ Mbps \tag{5.1}$$

and for the 176 × 144 pel images, the total bit rate for the uncoded video is

$$B_{uc} = (176 \times 144)\ pixels \times 12\ bits/pixel \times 30\ fps = 9\ Mbps \tag{5.2}$$

This is an enormous amount of data, which needs to be greatly compressed in order to be sent over bandwidth-constrained channels. There are two methods of performing this compression– constant quantization and constant bit rate. Sequences coded using a constant quantization step size will have roughly the same SNR of the reconstructed images but the bit rate will vary depending on the coding method. Similarly, sequences coded using a constant bit rate will have the same bit rate but the reconstructed image quality (and hence SNR) will vary depending on the coding method. The goal is to obtain a minimum bit rate for the sequences coded using constant quantization and a maximum SNR for the sequences coded using constant bit rate.

5.1 Constant Quantization Step Size

Each of these sequences was coded at a constant quantization step size (MPEG QP = 10) using NDME, encoder-based ME (EBME), and CR with 1 pel refinement. For all three coding methods, motion vectors were obtained at full pel accuracy and there was one motion vector generated for each macroblock. The average number of bits required to code each of the sequences using the different coding methods is shown

Table 5.1: Average number of kilobits per frame with fixed quantization step size.

Sequence Name	Encoder-Based Motion Estimation		Network Driven Motion Estimation		Conditional Replenishment	
Bus	58.6	(x1)	62.8	(x1.07)	98.1	(x1.67)
Flower	70.5	(x1)	73.2	(x1.04)	112.8	(x1.60)
Tennis	22.8	(x1)	27.6	(x1.21)	34.0	(x1.49)
Suzie	2.4	(x1)	2.6	(x1.08)	2.7	(x1.13)
Horse	23.5	(x1)	26.5	(x1.13)	46.9	(x2.00)
Gym	23.3	(x1)	27.7	(x1.19)	38.8	(x1.66)
Average	33.5	(x1)	36.7	(x1.10)	55.6	(x1.66)

in Table 5.1[1]. These results show that coding with NDME requires only a 10% higher average bit rate than coding with ideal encoder-based ME and achieves over a 33% decrease in average bit rate compared to coding with CR. Since the encoder performs the same computation for CR and NDME, this savings in bandwidth does *not* come at the expense of an increase in encoder power.

As described in Section 3.4, the decision about which mode to use to code the current frame is based on the variances of the CR and predicted motion compensated (PMC) difference images of the previous reconstructed frame. Figure 5-1a shows these variances for the "Bus" sequence. After the initial frames, which must be coded using intra-coding/CR, the variance of the PMC difference image is substantially lower than the variance of the CR difference image and the NDME system codes the remainder of the sequence using MP. Figure 5-1b shows the number of bits required to code each frame of the "Bus" sequence at a constant QP. The bit rate per frame of the NDME system coding algorithm is, on average, only 2.5-5 kbits worse than the ideal encoder-based ME system and 25-50 kbits *better* than the CR system. By coding this sequence with NDME rather than CR, the bandwidth decreases by as much as 50% with no increase in encoder power dissipation.

Similar results were obtained for both the "Flower" and "Horse" sequences, shown in Figures 5-2 and 5-3. Both these sequences contain large amounts of scene motion and are well suited for motion prediction. After the initial frames, the variance of the predicted motion compensated difference image is much less than the variance of the conditional replenishment difference image, so NDME determines that the remainder of these sequences is coded using MP. Figures 5-2b and 5-3b show the effectiveness of NDME in compressing

[1]For encoder-based ME, this number includes the coded MVs as well as the coded motion compensated difference image; for NDME, this number includes the coded predicted MVs (sent from the base-station to the encoder) and the MV refinements and the coded motion compensated difference image (sent from the encoder to the base-station); for CR with 1 pel refinement, this number includes the zero-vector refinements and the coded motion compensated difference image.

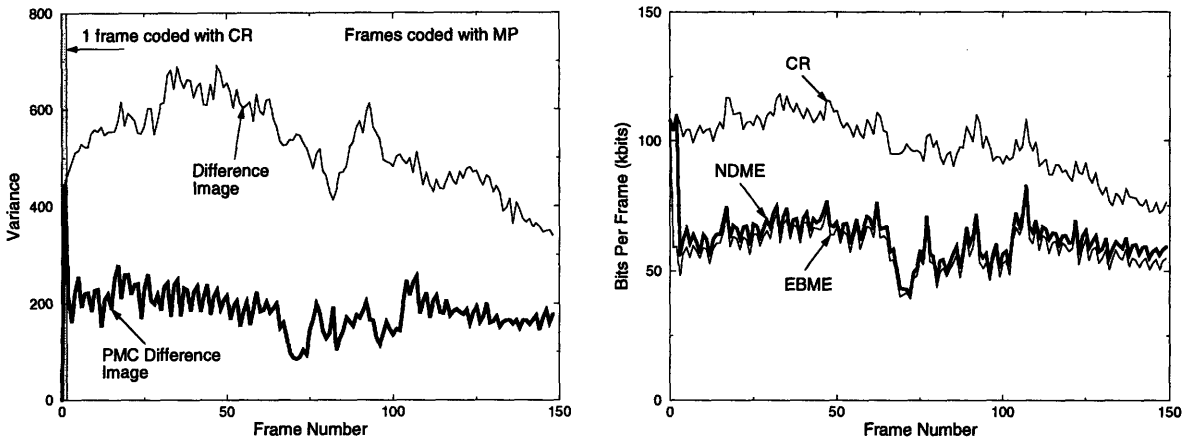


Figure 5-1: Bus Sequence. (a) Variance of CR and PMC difference images. (b) Number of bits required for each frame with fixed QP.

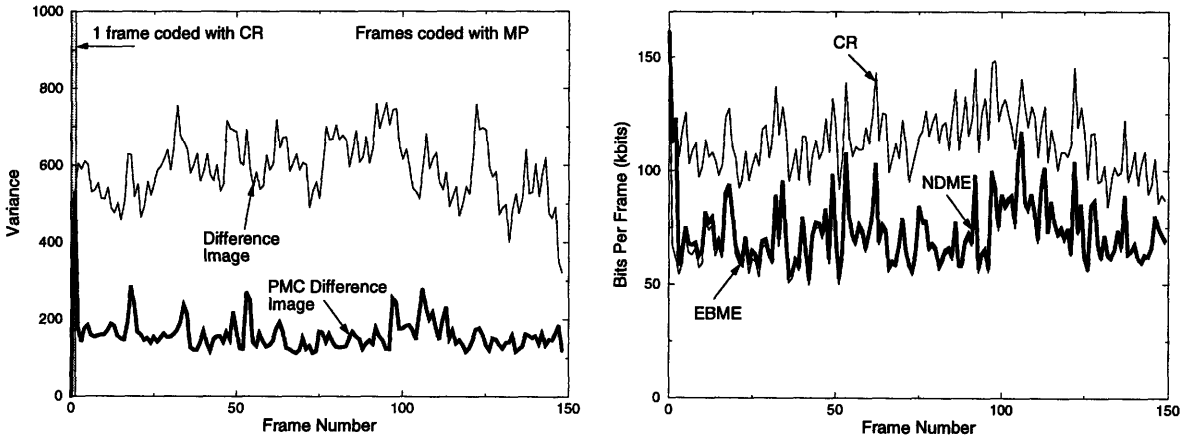


Figure 5-2: Flower Sequence. (a) Variance of CR and PMC difference images. (b) Number of bits required for each frame with fixed QP.

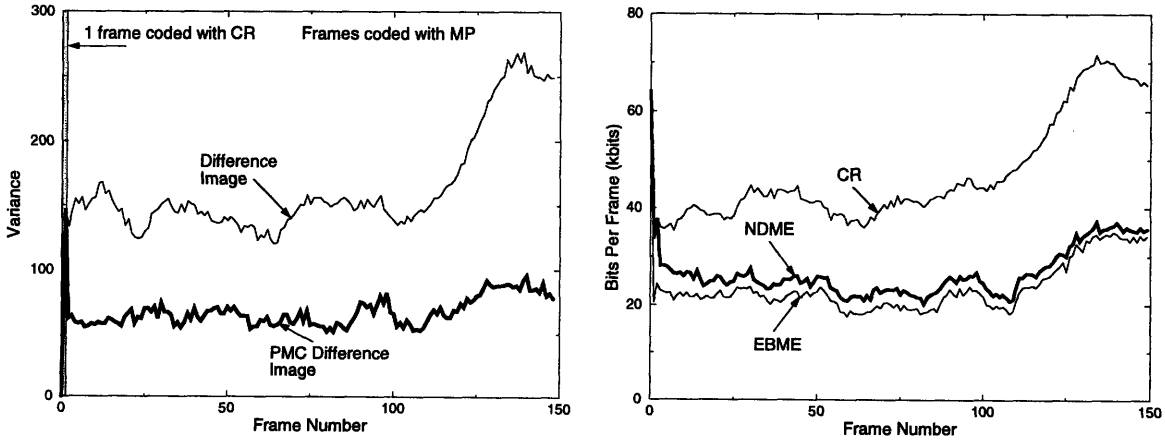


Figure 5-3: Horse Sequence. (a) Variance of CR and PMC difference images. (b) Number of bits required for each frame with fixed QP.

these images. For the "Flower" sequence, the compression ratio for NDME is almost exactly the same as the compression ratio for encoder-based ME and much greater than the compression ratio for CR. NDME requires 0-5 kbits more per frame than encoder-based ME and 25-50 kbits less per frame than CR. For the "Horse" sequence, NDME requires 2.5-5 kbits more per frame than encoder-based ME and 20-60 kbits less per frame than CR. These sequences illustrate the dramatic improvements in coding obtained using NDME rather than CR.

Comparatively, the "Tennis", "Suzie", and "Gym" sequences have much less motion. This can be seen by comparing the variances of the CR and PMC difference images, shown in Figure 5-4a for the "Tennis" sequence. For the first few frames, the variance of the predicted motion compensated difference image is very similar to the variance of the CR difference image. In this case, the NDME system preferences using the CR mode since no motion vectors need to be transmitted to the encoder. Scene motion begins around frame 25, which can be seen by the large increase in the variance of the CR difference image. At this point, it is advantageous to code the images using MP, so the NDME system switches to this mode. Around frame 90 there is a scene change and the encoder automatically sends an intra frame and forces the system into the CR mode. The remainder of the sequence has little motion, so the system remains in the CR mode. By adaptively switching between CR and MP, the NDME system is able to obtain bit rates as low as the CR system for frames with little or no motion and substantially lower than the bit rates from the CR system for frames with large amounts of motion. Figure 5-4b shows the number of bits required to code each frame of the "Tennis" sequence. Only the middle frames of the sequence are coded using MP; for these frames, the NDME system codes the images using approximately 5-10 kbits more than the ideal encoder-based ME system and 15-25 kbits fewer than the CR system.

Similar results were obtained for the "Suzie" and "Gym" sequences, shown in Figures 5-5 and 5-6. For the "Suzie" sequence, there is little motion in the beginning and the NDME system codes these initial frames using CR. As the scene motion increases, the system switches to MP, and when scene motion dies out, the system once again codes the frames using MP. The "Gym" sequence is coded using CR for the beginning segment, which contains little scene motion. The system switches from MP to CR followed again by MP depending on which mode will produce the most compression. This sequence shows the ability of the system to adaptively switch coding modes in order to obtain the greatest possible amount of compression for each frame.

5.2 Constant Bit Rate

The sequences were also coded with a constant bit rate (using the rate control module described in Section 4.4) to see the effects of these different coding algorithms on reconstructed image quality. Each of these sequences was coded at a constant bit rate using NDME, encoder-based ME, and CR with 1 pel refinement. For all three coding methods, motion vectors were obtained at full pel accuracy and there was one motion vector generated for each macroblock. Table 5.2 shows the average SNR of the reconstructed images using the

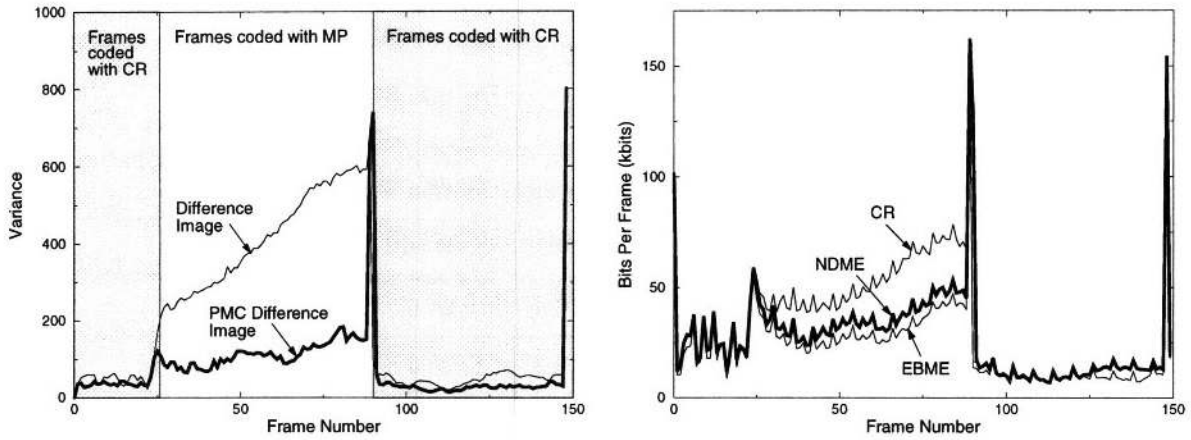


Figure 5-4: Tennis Sequence. (a) Variance of CR and PMC difference images. (b) Number of bits required for each frame with fixed QP.

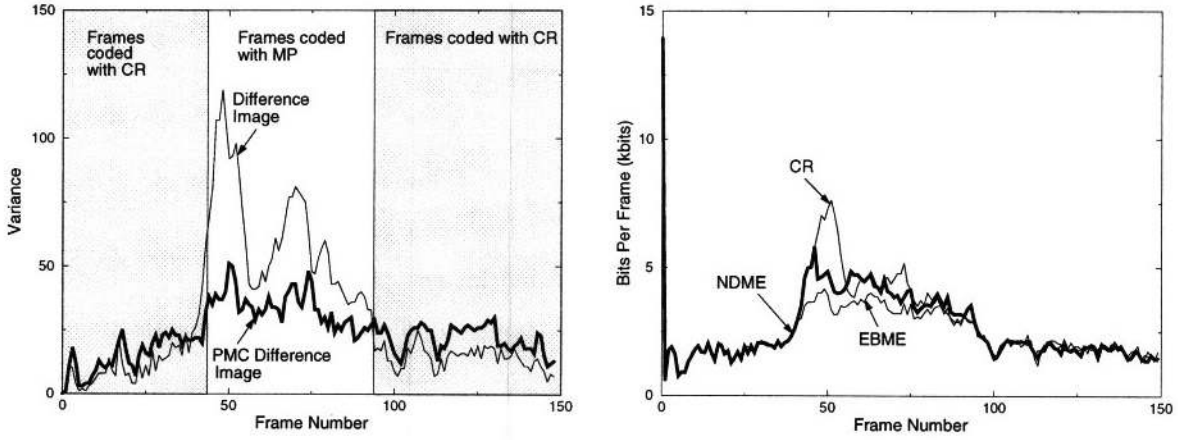


Figure 5-5: Suzie Sequence. (a) Variance of CR and PMC difference images. (b) Number of bits required for each frame with fixed QP.

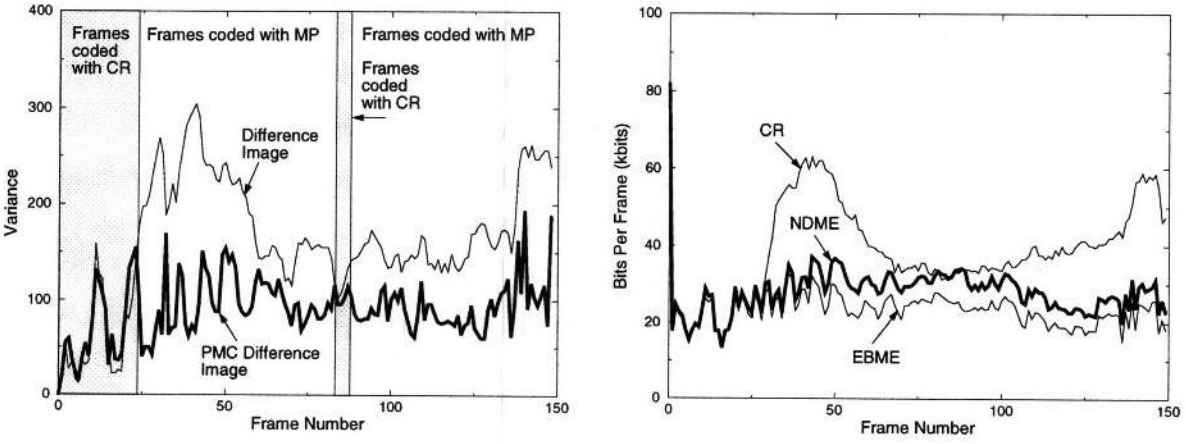


Figure 5-6: Gym Sequence. (a) Variance of CR and PMC difference images. (b) Number of bits required for each frame with fixed QP.

Table 5.2: Average SNR of reconstructed images and average quantization step size for constant bit rate coding.

Sequence Name	Bit Rate (Compression Ratio)	Encoder-Based Motion Estimation		Network Driven Motion Estimation		Conditional Replenishment	
		SNR (dB)	QP	SNR (dB)	QP	SNR (dB)	QP
Bus	1 Mbps (30:1)	25.6	16.4	24.6	18.0	22.0	24.9
Flower	1 Mbps (30:1)	23.9	19.2	23.3	20.2	20.3	29.0
Tennis	500 kbps (60:1)	29.8	12.8	28.7	14.6	26.0	18.2
Suzie	100 kbps (90:1)	32.3	8.7	32.7	9.5	32.3	9.8
Horse	500 kbps (60:1)	29.1	13.2	27.6	15.0	25.0	21.3
Gym	500 kbps (60:1)	29.5	13.7	27.6	16.7	25.9	20.3
Average	600 kbps (45:1)	28.4	14.0	27.4	15.7	25.3	20.6

three different coding methods for the 30fps sequences "Bus", "Flower", "Tennis", "Suzie", "Horse", and "Gym" and a constant bit rate (specified in the table)[2]. This table also shows the average quantization step size, which is related to the SNR of the reconstructed images. If the image to be coded (i.e., the motion compensated difference image) contains a large amount of information, the quantization step size will need to be increased in order to keep the bit rate constant for each image (or group of images). When the images are quantized coarsely, which occurs when the quantization step size is large, the reconstructed images show the quantization effects and produce low SNR values. Table 5.2 shows that the images coded using conditional replenishment have an average quantization step size of 20.6, which is approximately 25% larger than the average quantization step size when the sequences are coded using NDME, and the average quantization step size for NDME is increased by only 11% over the average quantization step size for encoder-based ME. Similarly, the average SNR for the sequences coded using CR is approximately 8% lower than the average SNR for the sequences coded using NDME, which is approximately 3% lower than the SNR for the sequences coded using encoder-based ME. These numbers show a quantitative measurement of the improvement in reconstructed image quality obtained using NDME rather than CR and the relatively small degradation in reconstructed image quality obtained using NDME rather than encoder-based ME. Thus there is very little image quality loss in order to realize the substantial encoder power reduction obtained using NDME rather than encoder-based ME.

[2]Different sequences are coding using different bit rates in order to best illustrate the quantization effects. For example, the simpler sequences, such as "Horse", require a lower average bit rate to make the quantization effects visible. Similarly, the images in the "Suzie" sequence are half the size of the images in the other sequences so this sequence requires a much lower bit rate than the other sequences.

These results can also be seen in Figures 5-7 to 5-12. Figure 5-7a shows the variance of the difference image and the predicted motion compensated difference image used to determine the coding mode for the "Bus" sequence coded at a constant bit rate of 1 Mbps. The variance of the predicted motion compensated difference image is substantially lower than the variance of the difference image for all frames following the initial startup frames. Thus the entire sequence (except the initial frames) is coded using motion prediction. Figure 5-7b shows the SNR for each reconstructed frame of this sequence. The SNR per frame of the reconstructed frames coded using NDME is approximately 1.5 to 3 dB higher than the SNR per frame of the reconstructed frames coded using CR and approximately 0.5 to 1.5 dB lower than the SNR per frame of the reconstructed frames coded using encoder-based ME. Thus NDME achieves a large improvement in the SNR of the reconstructed frames compared with CR for the "Bus" sequence.

The "Flower" sequence shows an even more dramatic improvement in SNR using NDME rather than CR (Figure 5-8). Once again, the variance of the predicted motion compensated difference image is much less than the variance of the difference image for all frames after the initial startup frames, and the sequence is coded using motion prediction. For this sequence, there is a 2.5 to 4 dB gain in SNR using NDME rather than CR and only a 0.5 to 1 dB loss in SNR using NDME rather than encoder-based ME. The "Horse" sequence (Figure 5-9) is also coded using MP for all frames. For this sequence, there is between 2 and 4 dB gain in SNR using NDME rather than CR and a 1 to 2 dB loss in SNR using NDME rather than encoder-based ME. Thus the reconstructed image quality will be greatly improved by using NDME rather than CR for these sequences.

The "Tennis", "Suzie", and "Gym" sequences show the advantage of switching between CR and MP when there is little or no scene motion. The beginning 25 frames of the "Tennis" sequence have very little motion, so the NDME system codes these frames using CR. Scene motion begins around frame 26, as is shown by the large increase in the variance of the difference image in Figure 5-10a. For these frames, the NDME system uses the MP mode. There is a scene change at Frame 89. This is detected by the encoder, which forces an intra-coded frame followed by a frame coded using CR. Since there is very little motion after the scene change, the NDME system remains in the CR mode. Figure 5-10b shows the SNR of the reconstructed images coded using NDME, CR, and encoder-based ME. The SNR of the NDME-coded frames is as much as 7 dB higher then the SNR of the corresponding frame coded using CR. This is because the CR system cannot detect the scene change, since the variance of the difference image at the scene change is not much greater than the variance of the difference images before the scene change. This is another advantage of using NDME rather than CR and is described in detail in Section 4.3. Since the CR system does not detect the scene change, it does not begin the new scene with an intra-coded frame. This causes the SNR of the reconstructed frames using NDME to be much higher than the SNR of the reconstructed frames using CR. However, even before the scene change, the SNR of the NDME system is approximately 2.5 dB higher than the SNR of the CR system for the frames coded using MP.

The "Suzie" sequence shows much less improvement in SNR using NDME rather than CR (Figure 5-

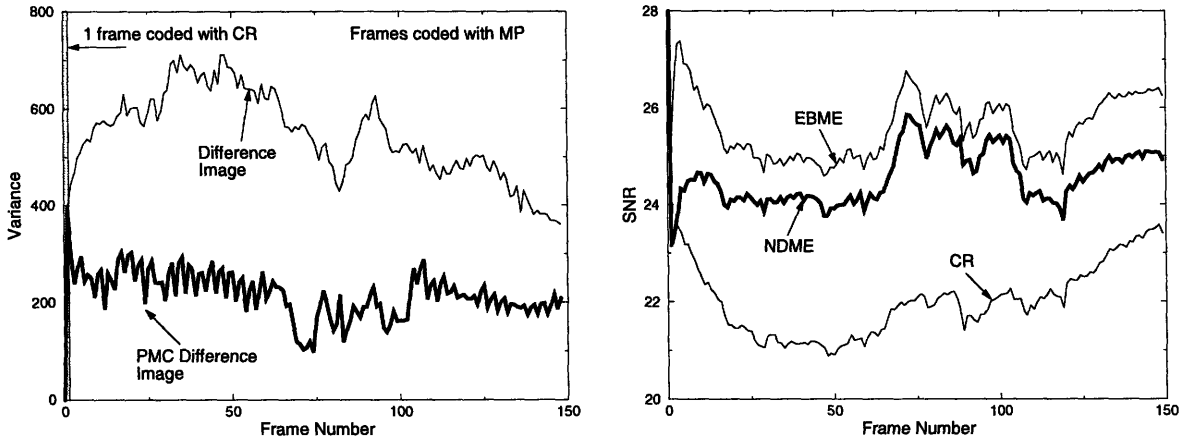


Figure 5-7: Bus Sequence. (a) Variance of CR and PMC difference images. (b) SNR per frame when coding the sequence at 1 Mbps.

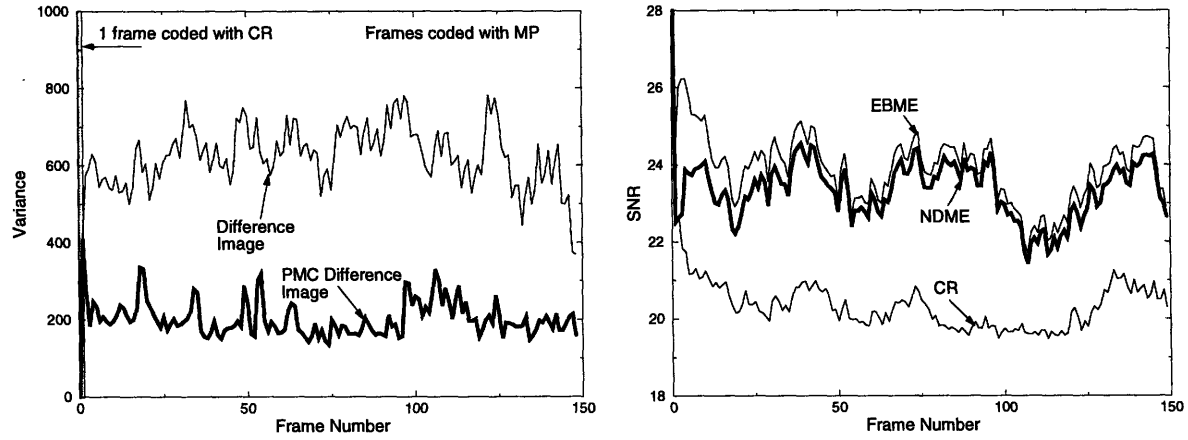


Figure 5-8: Flower Sequence. (a) Variance of CR and PMC difference images. (b) SNR per frame when coding the sequence at 1 Mbps.

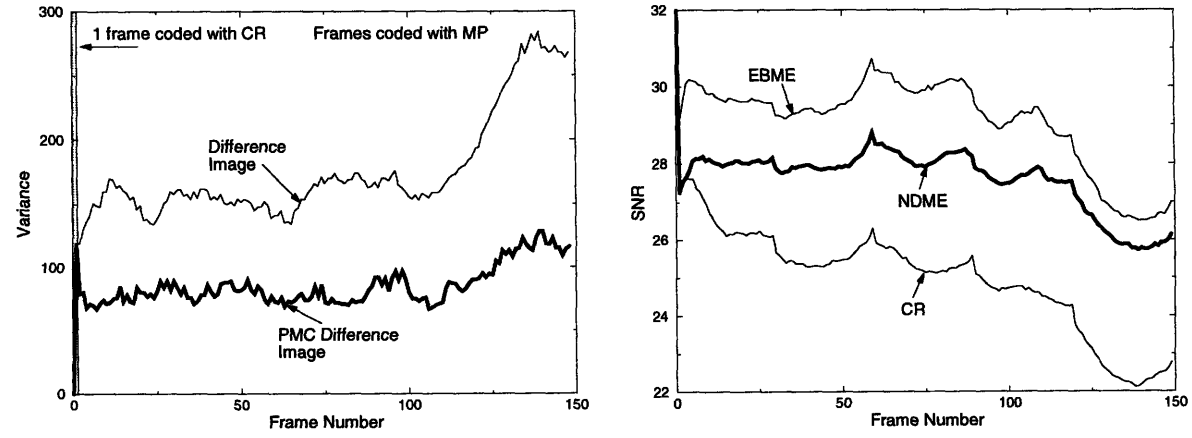


Figure 5-9: Horse Sequence. (a) Variance of CR and PMC difference images. (b) SNR per frame when coding the sequence at 500 kbps.

11). Only the middle few frames are coded using motion prediction, and for these frames, the increase in SNR is only about 1.5 dB over the frames coded using CR and about 1.5 to 3 dB worse than the frames coded using encoder-based ME. However, as will be shown in Section 5.2.1, the image quality of the frames coded using MP is much better than the image quality of the frames coded using CR and not much worse than the image quality of the frames coded using encoder-based ME. Thus the SNR does not necessarily accurately describe the degree of improvement in reconstructed image quality. The "Gym" sequence, shown in Figure 5-12, is coded using CR for the beginning frames of the sequence and MP for the images after scene motion begins (around frame 25). There is a large range of SNR improvement using NDME rather than CR for this sequence, from as little as 1 dB to as much as 4 dB. Similarly, the SNR of frames coded using NDME is between 1 dB and 4 dB lower than the frames coded using encoder-based ME.

5.2.1 Reconstructed Images

Sample reconstructed frames from the "Tennis" sequence are shown in Figure 5-13. These images show the relatively small degradation in image quality obtained by using NDME rather than encoder-based ME and the large improvement in image quality obtained by using NDME rather than CR for this frame. Sample reconstructed frames from each of the other sequences are shown in Appendix B (Figures B-6 to B-10). The quantization effects are even more apparent when the images are viewed in motion. The background appears to jump around as a result of the large quantization of the images coded using CR, whereas these quantization effects are not as apparent in the sequences coded using NDME. The video tape included with this thesis shows these sequences in motion.

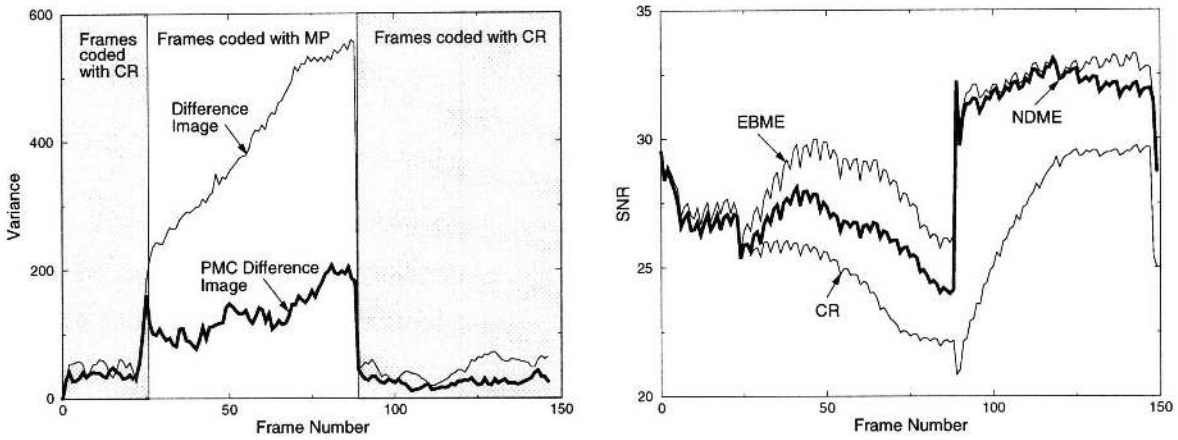


Figure 5-10: Tennis Sequence. (a) Variance of CR and PMC difference images. (b) SNR per frame when coding the sequence at 500 kbps.

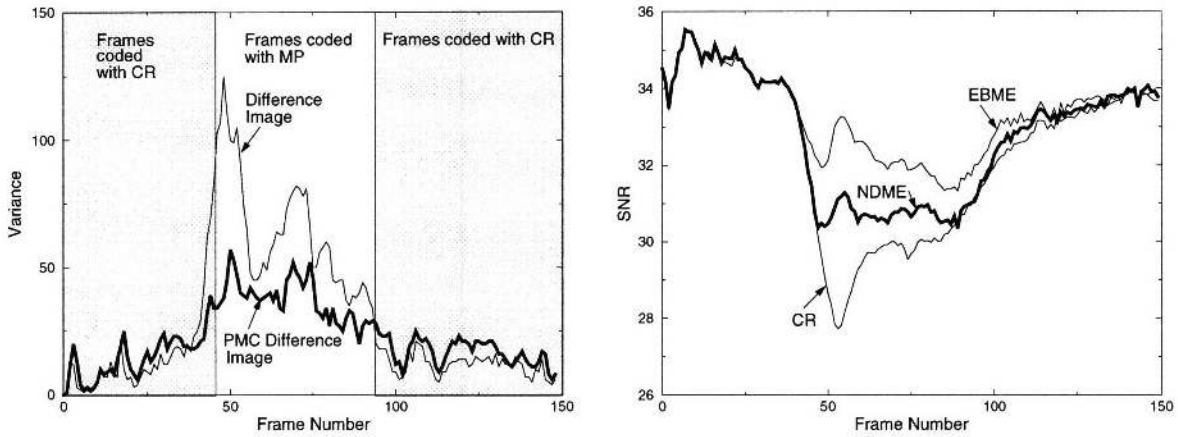


Figure 5-11: Suzie Sequence. (a) Variance of CR and PMC difference images. (b) SNR per frame when coding the sequence at 100 kbps.

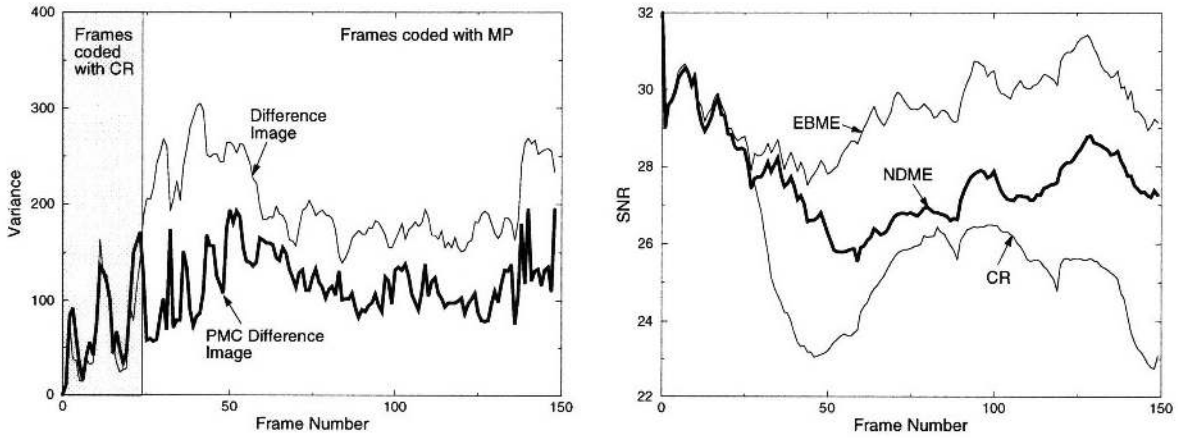


Figure 5-12: Gym Sequence. (a) Variance of CR and PMC difference images. (b) SNR per frame when coding the sequence at 500 kbps.

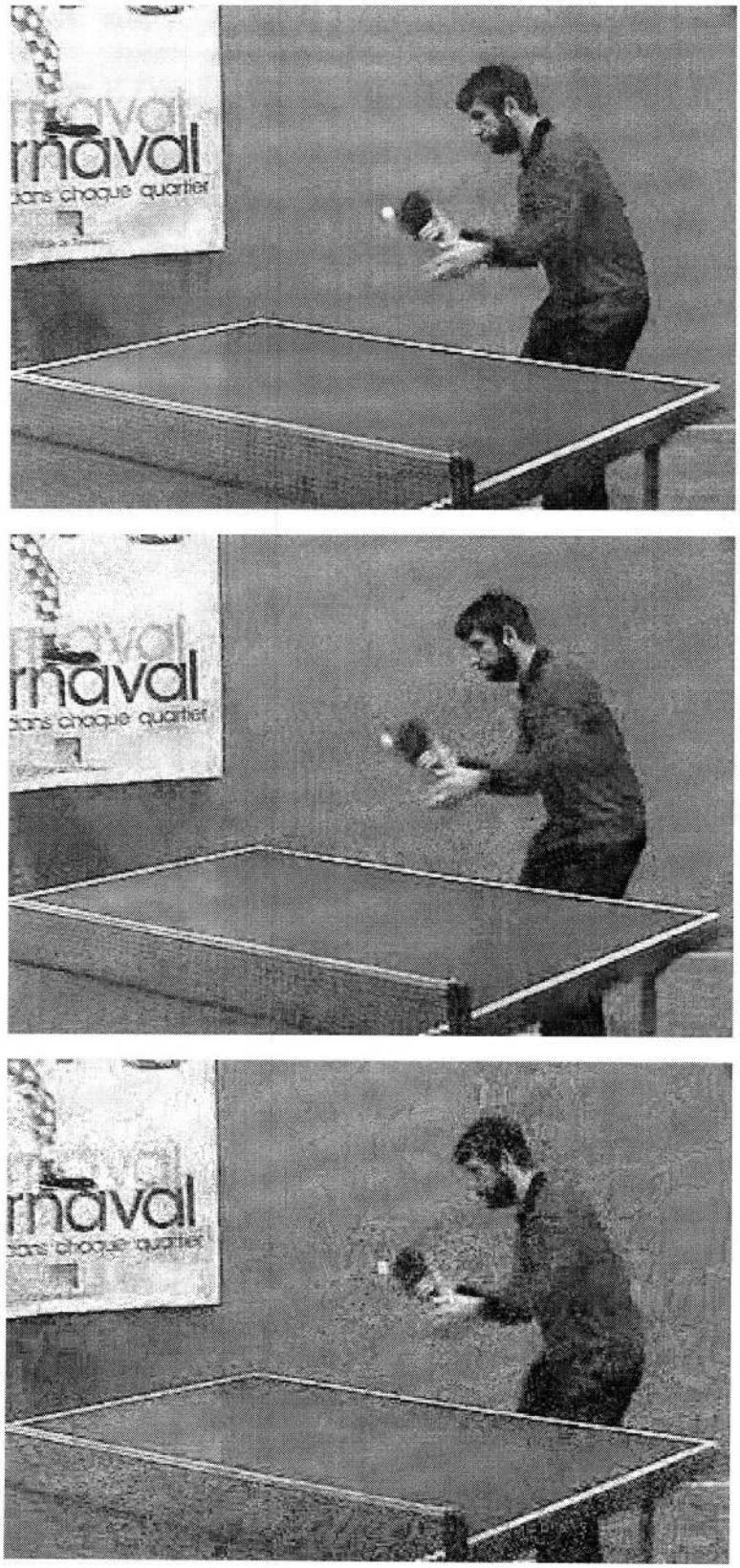


Figure 5-13: Tennis sequence coded with constant bit rate. (a) Encoder-based ME, SNR = 27.3 dB (b) Network driven ME, SNR = 25.4 dB (c) Conditional replenishment, SNR = 22.3 dB

Chapter 6

Conclusion

Video is becoming an integral part of many portable devices, such as wireless cameras, personal communication devices, and video cellular phones. Due to the massive amount of data contained in video signals and the limited bandwidth of the wireless channel, compression techniques for these applications are extremely important. However, most algorithms which produce substantial compression are computationally intense. For portable video devices, this computational complexity translates into a shorter battery lifetime. If, on the other hand, computationally simple compression algorithms are used, there may not be an acceptable amount of compression of the video signal. This translates into using a larger quantization step size in order to compress the data to fit into the limited bandwidth of the channel, which causes the quality of the decoded images to be degraded. This illustrates the tradeoff between encoder computation and amount of compression when motion estimation is performed at the video encoder.

This tradeoff is minimized in network driven motion estimation because the ± 16 pel motion estimation search is *removed* from the encoder and performed at a high powered server on the network. Figure 6-1 shows a flow graph of the entire network driven motion estimation algorithm. The base-station (or a remote high powered server on the wired network) performs the computationally intense search for each frame's predicted motion vectors, whereas the encoder only performs a local search to refine the predicted motion vectors. Thus the encoder search window has $w = 3$ and $h = 3$ (1 pel refinement), which requires only $2wh = 18$ ops/pixel. Comparing this with the number of ops/pixel of the various motion estimation algorithms outlined in Table 2.1 shows that network driven motion estimation requires many fewer computations compared with most of these motion estimation algorithms. In addition, the results shown in Chapter 5 illustrate that network driven motion estimation produces fairly to highly accurate motion vectors. Full search motion estimation has a search window of size 33×33 and requires $2wh = 2178$ ops/pixel. Thus network driven motion estimation reduces the encoder computation by a factor of 120 compared with full search motion estimation, which represents a power reduction of over *two orders of magnitude*. The results of the experiments show that network driven motion estimation can obtain this substantial reduction in encoder power with only a small increase in data bandwidth. Alternatively, the network driven motion

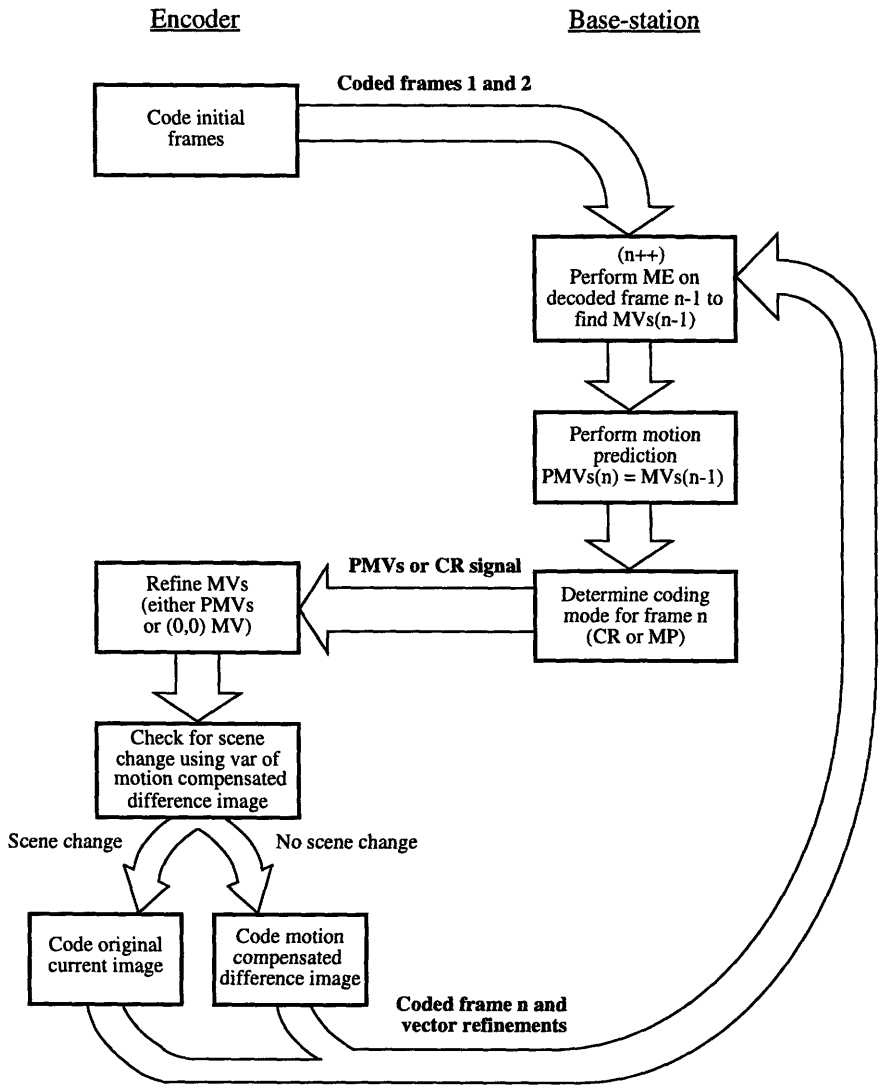


Figure 6-1: Flow graph showing the network driven motion estimation algorithm.

estimation system uses the same amount of power as the CR system while obtaining a significant decrease in data bandwidth. By transferring the majority of the computation required for motion estimation from the power limited portable video terminals to a high powered system on the network, network driven motion estimation can produce low bandwidth coded video using minimal power consumption.

6.1 Future Work

All the results in this thesis compared NDME to full search encoder-based ME and CR with one pel refinement. It would be interesting to see how NDME compares to some of the fast motion estimation algorithms described in Section 2.5. A full comparison showing bandwidth of the coded video versus amount of computation for the motion estimation algorithm would give designers of wireless video systems a better idea of

which algorithm is appropriate for the particular application. For example, one of the fast motion estimation algorithms may require extra computation or may not produce as high compression ratios as NDME but does not require a low latency reverse link between the base-station and the encoder and may thus be preferable to NDME. Determining a complete set of advantages and disadvantages for different motion estimation algorithms will allow the designer to choose the optimum algorithm given the application constraints.

In Section 4.1, it was found that increasing the number of motion vectors per image actually decreases the total bit rate of the system. To complete this experiment, the number of motion vectors per image should be increased beyond one motion vector per macroblock to four motion vectors per macroblock (or 1320 motion vectors per image for the CIF images). This is the same as using block motion estimation, which is part of the MPEG standard (where there are 4 blocks per macroblock). If the total bit rate decreases using one motion vector per block, this may be preferable. However, this quadruples the reverse channel bandwidth, because four times as many predicted motion vectors need to be sent from the base-station to the encoder. This may be prohibitive for certain systems, especially if there is not a significant gain in total bandwidth by using one motion vector per block rather than one motion vector per macroblock.

The relationship between frame variance and bit rate described in Section 3.4 needs to be explored further. By finding out how these two parameters are related (or if, in fact, there is not a relationship between them), the algorithm which determines the coding mode can be optimized, rather than using the empirical results based on a few test sequences. Similarly, the scene change detector can be optimized by studying the relationship between variance and scene content. Rather than using the absolute value of the variance of the motion compensated image to determine if a scene change has occurred, the difference between this variance and the variance of the previous motion compensated difference image might be more suitable, since the variance rises sharply when a scene change occurs. Perhaps an entirely different scene change detector algorithm might be more accurate or use less encoder computation.

Section 4.4 stated that NDME requires a constant sampling rate between frames. However, a system could be designed to use the H.263 rate control, which performs temporal quantization and drops frames when the allocated bit rate gets too low, if the encoder can let the base-station know how many frames will be dropped. The base-station can then either extrapolate the predicted motion vectors for the specified number of frames, or it can compute new predicted motion vectors using the frames that are at the specified number of frames in the past to predict the next frame's motion vectors.

By optimizing these algorithms, the NDME system may produce even better results than the ones discussed in Chapter 5. Comparing the bit rate (or the SNR of the reconstructed images) of sequences coded using network driven motion estimation with sequences coded using some of the fast motion estimation algorithms in addition to full search motion estimation and conditional replenishment will confirm that network driven motion estimation is a desirable algorithm for performing motion estimation in battery-operated portable video devices.

Appendix A

Tables

Table A.1: Huffman table used to compress the predicted motion vector values.

Value	Codeword	Value	Codeword
0	1		
1	010	-1	011
2	0010	-2	0011
3	00010	-3	000111
4	0000110	-4	0000111
5	00001010	-5	00001011
6	00001000	-6	00001001
7	00000110	-7	00000111
8	00010110	-8	00010111
9	0000010100	-9	0000010101
10	0000010010	-10	0000010011
11	00000100010	-11	00000100011
12	00000100000	-12	00000100001
13	00000011110	-13	00000011111
14	00000011100	-14	00000011101
15	00000011010	-15	00000011011
16	00000011000	-16	00000011001
17	00000010110	-17	00000010111
18	00000010100	-18	00000010101
19	00000010010	-19	00000010011
20	00000010000	-20	00000010001
21	00000001110	-21	00000001111
22	00000001100	-22	00000001101
23	00000001010	-23	00000001011
24	00000001000	-24	00000001001
25	000000001110	-25	000000001111
26	000000001100	-26	000000001101
27	000000001010	-27	000000001011
28	000000001000	-28	000000001001
29	000000000110	-29	000000000111
30	000000000100	-30	000000000101
31	0000000000110	-31	0000000000111
32	0000000000100	-32	0000000000101

Table A.2: Huffman table used to compress the motion vector refinement values.

Value	Codeword	Value	Codeword
0	1		
1	01	-1	00

Appendix B

Figures

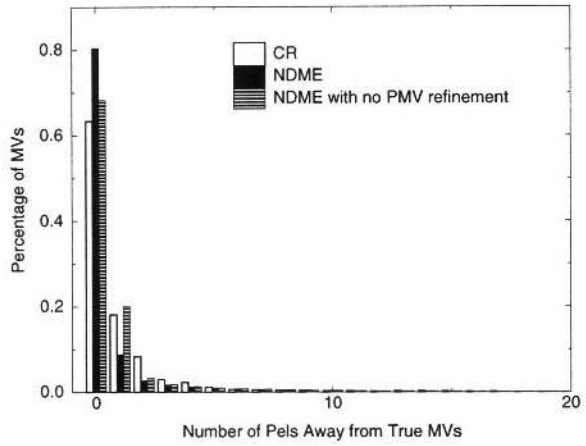


Figure B-1: Flower Sequence. Distance between 1. CR MVs 2. NDME MVs 3. unrefined PMVs and true MVs.

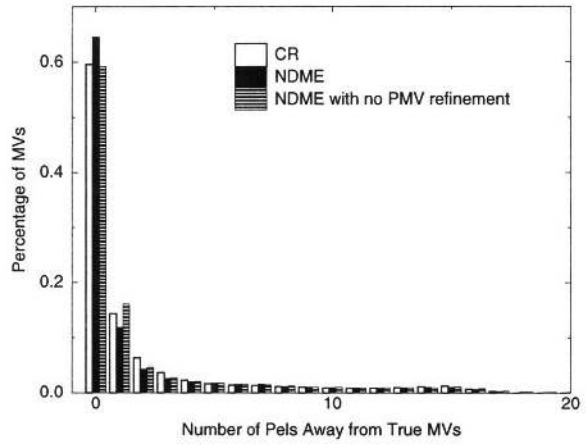


Figure B-2: Tennis Sequence. Distance between 1. CR MVs 2. NDME MVs 3. unrefined PMVs and true MVs.

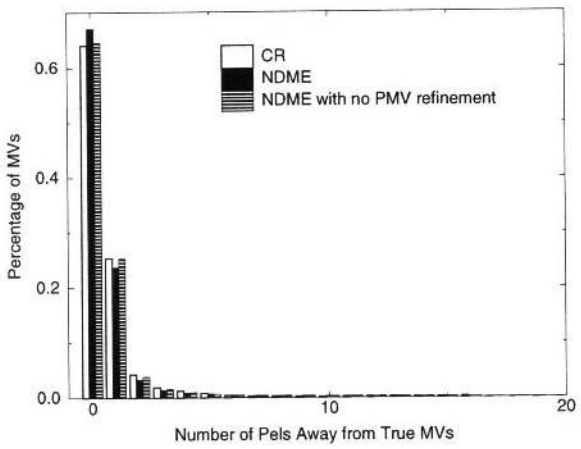


Figure B-3: Suzie Sequence. Distance between 1. CR MVs 2. NDME MVs 3. unrefined PMVs and true MVs.

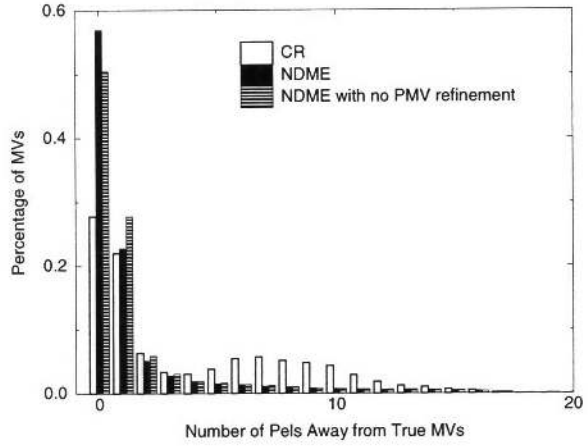


Figure B-4: Horse Sequence. Distance between 1. CR MVs 2. NDME MVs 3. unrefined PMVs and true MVs.

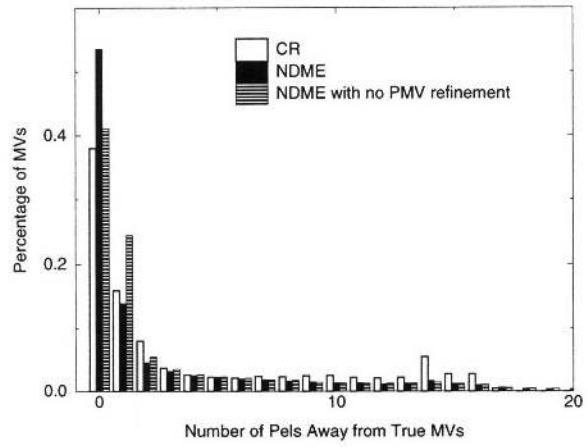


Figure B-5: Gym Sequence. Distance between 1. CR MVs 2. NDME MVs 3. unrefined PMVs and true MVs.

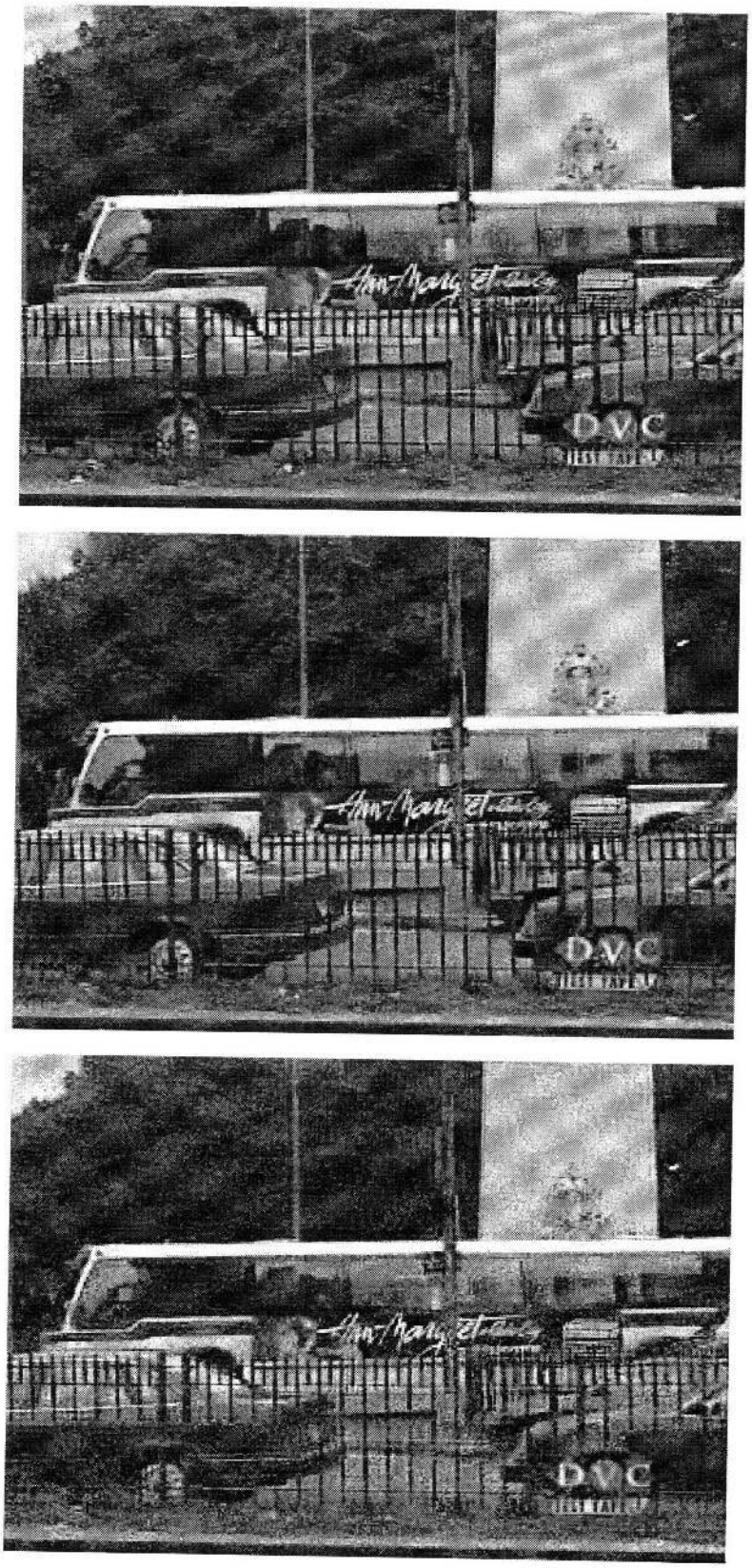


Figure B-6: Bus sequence coded with constant bit rate. (a) Encoder-based ME, SNR = 27.3 dB (b) Network driven ME, SNR = 26.6 dB (c) Conditional replenishment, SNR= 22.9 dB

Figure B-7: Flower sequence coded with constant bit rate. (a) Encoder-based ME, SNR = 25.1 dB (b) Network driven ME, SNR = 24.7 dB (c) Conditional replenishment, SNR = 21.2 dB

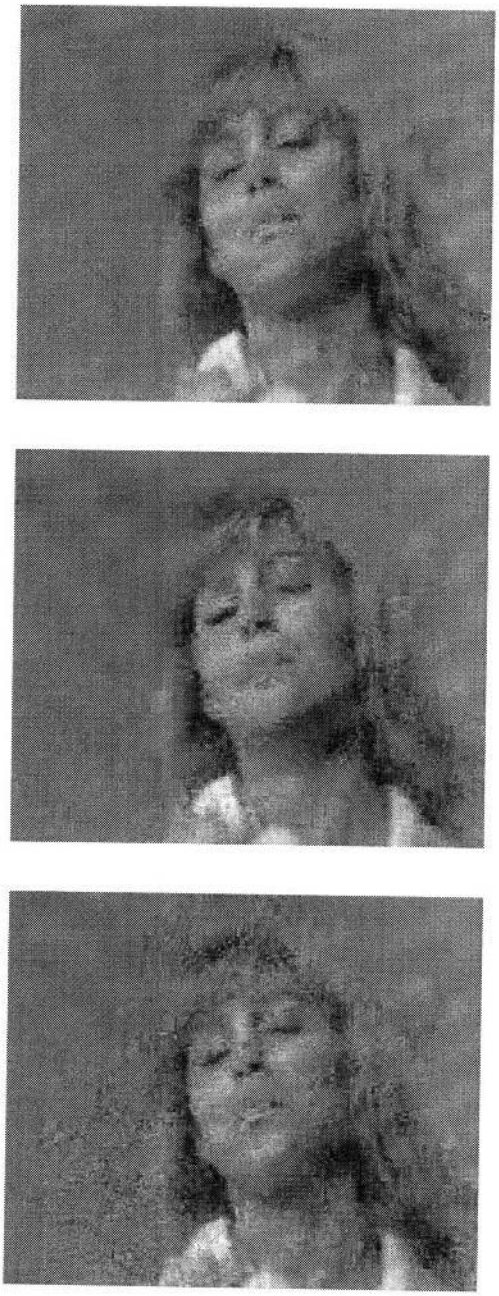


Figure B-8: Suzie sequence coded with constant bit rate. (a) Encoder-based ME, SNR = 32.4 dB (b) Network driven ME, SNR = 30.2 dB (c) Conditional replenishment, SNR= 27.5 dB

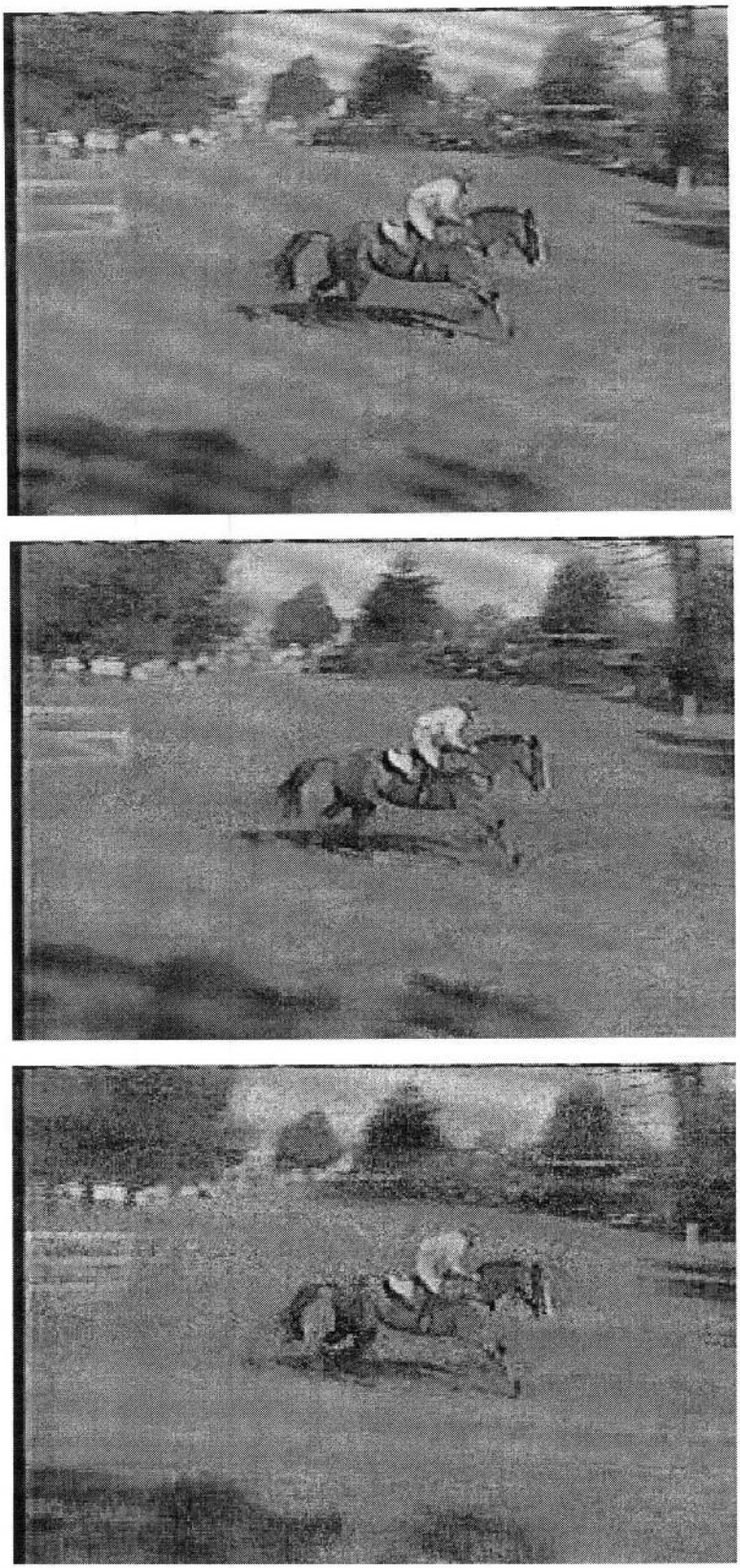


Figure B-9: Horse sequence coded with constant bit rate. (a) Encoder-based ME, SNR = 29.9 dB (b) Network driven ME, SNR = 28.0 dB (c) Conditional replenishment, SNR = 25.5 dB

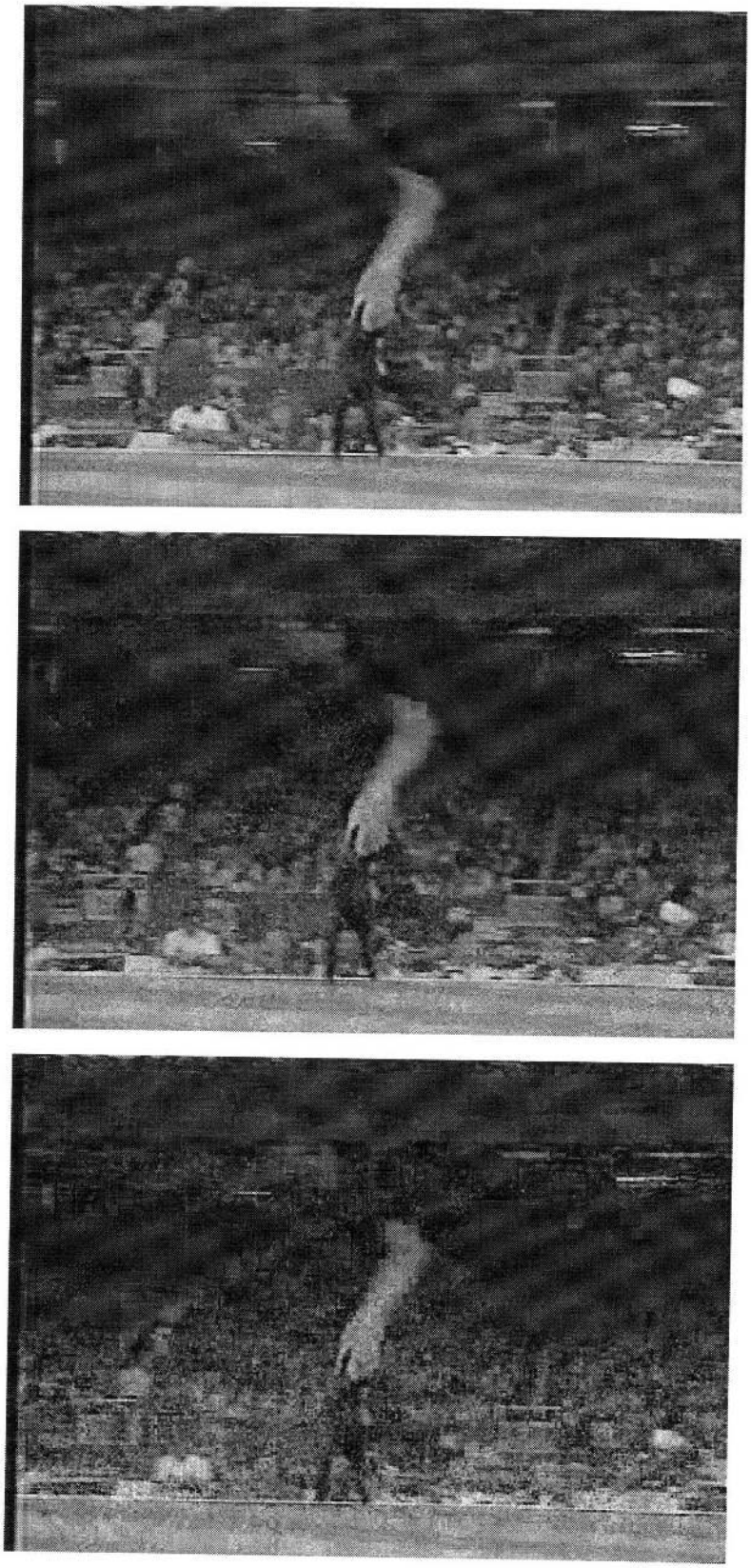


Figure B-10: Gym sequence coded with constant bit rate. (a) Encoder-based ME, SNR = 30.8 dB (b) Network driven ME, SNR = 28.2 dB (c) Conditional replenishment, SNR = 25.0 dB

Appendix C

Global Motion Estimation Algorithms

C.1 The Most Common Motion Vector Algorithm

The MCMV algorithm is a method of global motion estimation that determines which local MV is used most often and chooses this vector as the global motion vector. This global motion estimation method assures that the maximum number of macroblocks will achieve their minimum SAD value.

$$\vec{MV}_{global} = \max_{g\vec{m}v}(\sum_{i=1}^{N}[(\vec{MV}_i == g\vec{m}v) \ ? \ 1 \ : \ 0 \ ]) \tag{C.1}$$

where $\vec{MV}_i$ is the local MV for the i^{th} macroblock and N is the total number of macroblocks in the image. This method will achieve good compression only if the majority of MVs are the same, since it may produce very high energy residuals for macroblocks whose MVs do not match the global MV.

C.2 The Tse and Baker Global Motion Estimation Algorithm

The Tse and Baker global motion estimation algorithm [14] models the global motion as a combination of pan (translational movement) and zoom (change of focal length). This algorithm assumes that the global motion is the dominant motion in the scene and the motion of local objects is uncorrelated noise. The global motion parameters (p_x, p_y, and f_z) are estimated from the local MVs ($\vec{MV}_i$). The local motion of a pixel can be described by

$$\vec{U}_i = \vec{U}'_i + \vec{MV}_i \tag{C.2}$$

where $\vec{U}_i$ is the location of the pixel in the current frame and $\vec{U}'_i$ is the location of the pixel in the previous frame. Using the global motion parameters, the pixel motion is described by

$$\widehat{\vec{U}}_i = f_z\vec{U}'_i + \vec{p} \tag{C.3}$$

where $\widehat{\vec{U}}_i$ is an estimate of the globally motion compensated location of the pixel in the previous frame. The estimated global displacement vectors are given by

$$\widehat{\vec{MV}}_i = \widehat{\vec{U}}_i - \vec{U}'_i = (f_z - 1)\vec{U}'_i + \vec{p} \tag{C.4}$$

The goal of this algorithm is to obtain the maximum number of matches between the estimated global displacement vectors $\widehat{\vec{MV}}_i$ and the "true" local MVs $\vec{MV}_i$. This is accomplished by minimizing the l_2 norm of the MV displacement error

$$E = \sum_i \|\widehat{\vec{MV}}_i - \vec{MV}_i\|^2 \tag{C.5}$$

The authors in [14] show that this minimization results in the following solutions for the global zoom and pan parameters

$$f_z = \frac{\sum_i \langle \vec{U}_i, \vec{U}'_i \rangle - \frac{1}{N} \langle \sum_i \vec{U}_i, \sum_i \vec{U}'_i \rangle}{\sum_i \langle \vec{U}'_i, \vec{U}'_i \rangle - \frac{1}{N} \langle \sum_i \vec{U}'_i, \sum_i \vec{U}'_i \rangle} \tag{C.6}$$

$$\vec{p} = \frac{1}{N} (\sum_i \vec{U}_i - f_z \sum_i \vec{U}'_i) \tag{C.7}$$

where $\langle ., . \rangle$ denotes inner product, and N is the number of macroblocks used in the parameter estimation. The initial estimates use the local MVs from all the macroblocks in the image. Since local motion which differs from the global motion estimate may reduce the accuracy of the global motion parameters, the algorithm can be iterated to refine the estimates. For each iteration, only macroblocks whose local MVs match the current global motion estimates are used in determining the new parameter values. The same minimization procedure (Equations C.6 and C.7) is used for each refinement, with the exception that the summations are only over the macroblocks which have local motion consistent with the current estimate of the global motion. While this method does not guarantee that the global motion will be optimal for the maximum number of macroblocks, it does assure that the global motion estimate is close to the maximum number of local motion estimates. Thus, even though the minimum SAD for individual macroblocks may not be achieved, the overall SAD of the image should be minimized.

Appendix D

Running the NDME System Using Images From the Accom

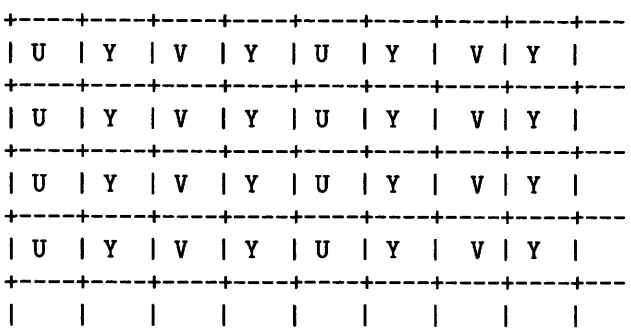
D.1 Acquisition of Video Material

The video sequences "Horse" and "Gym" were obtained by transferring video from a VCR tape to a digital video recorder (the Accom digital recorder). The video is stored in the Accom memory and can be transferred to a workstation using the rcp command. For example, the following C-shell script will retrieve 150 frames from the Accom memory starting at memory location 100 and store them on the workstation in YUV 4:2:2 format as horse.xxx.yuv[1]:

```
#! /bin/csh -f

# Read input arguments
set file = horse
@ start = 0
@ end = 149
@ accom_frame = 100

@ cur = $start
while ($cur <= $end)
  if ($cur < 10) then
    set c=00$cur
  else if ($cur < 100) then
    set c=0$cur
  else
    set c=$cur
  endif

  # transferring frame $accom_frame from the accom to $file.$c.yuv
  rcp accom:$accom_frame $file.$c.yuv
  @ cur++
  @ accom_frame++
end
```

The images horse.xxx.yuv are size 720×486 pixels and have the following representation, where each chrominance value (U and V) and luminance value (Y) is represented with a byte of data:

[1] xxx represents a three digit number from 000 to 149.



Thus there are twice as many Y components as U and V components (due to the 4:2:2 subsampling of the chrominance). In order to obtain images suitable as input to the NDME video system, these image files must be parsed into three separate files, horse.xxx.Y, horse.xxx.U, and horse.xxx.V, the size of the image must be reduced (for the "Horse" and "Gym" sequences, the images were reduced to 352×240 pixels), and the chrominance must be further subsampled to produce YUV in the 4:1:1 format. The following C code will produce the correct size and format images:

```
#include <stdio.h>
#include <stdlib.h>

void dumpimage(unsigned char *image, char filename[255], int tot_pels,
               int tot_lines, int pels, int lines)
{
  unsigned char temp_char;
  int row,col,i;
  unsigned char *out;
  FILE *outFile;
  int ppels, plines;

  outFile = fopen(filename,"w");
  out = image;
  ppels = (int) (tot_pels / pels);
  plines = (int) (tot_lines / lines);

  /* subsampling the image to the appropriate size */
  for (row = 0; row < lines; row++) {
    for (col = 0; col < pels; col++) {
      temp_char = (char) (*out++);
      fwrite(&temp_char,1,1,outFile);
      for (i = 1; i < ppels; i++)
        temp_char = (char) (*out++);
    }
    for (col = 0; col < tot_pels - pels*ppels; col++)
      temp_char = (char) (*out++);
    for (i = 1; i < plines; i++)
      for (col = 0; col < tot_pels; col++)
        temp_char = (char) (*out++);
  }

  fclose(outFile);
  return;
}
```

```
void SeperateImage(FILE *inFile, char *outfile, int frm_no, int tot_pels,
                int tot_lines, int pels, int lines)
{
  unsigned char temp_char;
  int row, col, i;
  char fname[256];
  unsigned char *lum, *cr, *cb;
  unsigned char *y, *u, *v;

  lum = (unsigned char *)malloc(sizeof(unsigned char)*tot_pels*tot_lines);
  cr = (unsigned char *)malloc(sizeof(unsigned char)*tot_pels*tot_lines/4);
  cb = (unsigned char *)malloc(sizeof(unsigned char)*tot_pels*tot_lines/4);

  /* Separate luminance and chrominance */
  y = lum; u = cr; v = cb;
  for (row = 0; row < tot_lines; row+=2) {
    for (col = 0; col < tot_pels; col+=2) {
      fread(&temp_char,1,1,inFile);
      *u++ = (int) temp_char;
      fread(&temp_char,1,1,inFile);
      *y++ = (int) temp_char;
      fread(&temp_char,1,1,inFile);
      *v++ = (int) temp_char;
      fread(&temp_char,1,1,inFile);
      *y++ = (int) temp_char;
    }
    for (col = 0; col < tot_pels; col+=2) {
      fread(&temp_char,1,1,inFile);
      fread(&temp_char,1,1,inFile);
      *y++ = (int) temp_char;
      fread(&temp_char,1,1,inFile);
      fread(&temp_char,1,1,inFile);
      *y++ = (int) temp_char;
    }
  }
  sprintf(fname, "%s.%03d.Y", outfile, frm_no);
  dumpimage(lum,fname,tot_pels,tot_lines,pels,lines);

  sprintf(fname, "%s.%03d.U", outfile, frm_no);
  dumpimage(cr,fname,tot_pels/2,tot_lines/2,pels/2,lines/2);

  sprintf(fname, "%s.%03d.V", outfile, frm_no);
  dumpimage(cb,fname,tot_pels/2,tot_lines/2,pels/2,lines/2);

  free(lum); free(cr); free(cb);
  return;
}

main (int argc,char *argv[])
{
  int start, end, i;
  int pels, lines;
  int tot_lines, tot_pels;
  char *outfile;
```

```
  FILE *inFile;

  argc--;
  argv++;
  if(argc != 5) {
    printf("USAGE: parse2_color pels lines image filename number\n");
    exit(0);
  }
  /* size of images to be input to NDME video system */
  pels = atoi(*argv++);
  lines = atoi(*argv++);

  /* size of the images from the accom */
  tot_pels = 720;
  tot_lines = 486;

  if (!(inFile = fopen(*argv++,"r"))) {
    printf("ERROR: Can't open image file for reading\n");
    exit(0);
  }
  outfile = *argv++;
  i = atoi(*argv++);
  SeperateImage(inFile, outfile, i, tot_pels, tot_lines, pels, lines);
  fclose(inFile);
}
```

Running this program on the image horse.xxx.yuv produces the files horse.xxx.Y, horse.xxx.U, and horse.xxx.V, where horse.xxx.Y represents the luminance component of the image and is the specified size (i.e., 352×240), and horse.xxx.U and horse.xxx.V represent the chrominance components of the image and are $\frac{1}{4}$ the size of the luminance image (i.e., 176×120).

D.2 Running the NDME Test System

Once the images are in the correct format, they can be compressed using the H.263 encoder with NDME extension. The name of the encoder is tmn, and in order to run the tmn program, the correct directories must be set. The directory "mvs" must be created to hold the motion vectors from the reconstructed previous frame; the directory "pred" must be created to hold the predicted motion vectors[2]; and the directory "refine" must be created to hold the refined PMVs. For every frame, there is a file added to each directory. For example, for frame 50 a file is written into the directory mvs (mv.050) which contains the motion vectors for the reconstructed frame 50; a file is written into the directory pred (pred.050) which contains the PMVs for frame 50 (the MVs for the reconstructed frame 49); and a file is written into the directory refine (rmv.050) which contains the refined PMVs for frame 50. An output file, which contains the reconstructed image, is also created for each image, and the compressed bits for the frame are added to the bit stream file. For example, the tmn program is run as follows:

```
tmn -a 0 -b 149 -i horse -B horse.stream -o horse.recon -c 3 >log.horse
```

This produces the compressed bit stream horse.stream and the output images horse.recon.xxx.Y, horse.recon.xxx.U, and horse.recon.xxx.V for all images from 0 to 149. All the statistics for each image, such as the number of bits per image, the SNR of each image, the variance of the motion compensated image, etc., as well as the total statistics for the sequence are stored in the file log.horse; this file is therefore very useful in obtaining results about the coding method.

[2]Note that for the motion vector prediction method used in the NDME system, the predicted motion vectors of the current frame are the same as the motion vectors of the reconstructed previous frame.

D.3 Decoding NDME Test System Bit Streams

There is a decoder for the H.263 bit stream which is run as follows:

```
tmndecode -o0 horse.stream horse.recon.%03d.yuv
```

The -o0 option saves the reconstructed images in files called horse.recon.xxx.Y, horse.recon.xxx.U, and horse.recon.xxx.V, which represent the YUV 4:1:1 format The other output options are -o1, which is SIF format, -o2, which is TGA format, -o3, which is PPM format, -o4, which displays the images as a movie using X11, and -o5, which is YUV concatenated into one file.

D.4 Displaying Video Material

Once the reconstructed images are created, they can be viewed in real time using the Accom. However, the images must be converted to the RGB format and either upsampled to 720×486 pixels or filtered to include a black border which will make the size of the image 720×486. The images can be converted from YUV to RGB using the following C code:

```
#include <stdio.h>
#include <stdlib.h>

FILE *inFile;
FILE *outFile;
int pels, lines;

unsigned char *loadimage(int width, int height)
{
  unsigned char temp_char;
  int i, row, col;
  unsigned char *image;
  unsigned char *res;

  image = (unsigned char *)malloc(sizeof(unsigned char)*width*height);
  res = image;
  for (row = 0; row < height; row++)
    for (col = 0; col < width; col++) {
      fread(&temp_char,1,1,inFile);
      if (feof(inFile)) {
        printf("ERROR: too few pixels\n");
        exit(0);
      }
      *res++ = temp_char;
    }
  fread(&temp_char,1,1,inFile);
  if (feof(inFile) == 0) {
      printf("ERROR: too many pixels\n");
      exit(0);
    }

  return(image);
}

void dumpimage(unsigned char *image, int cols, int rows) {
  unsigned char temp_char;
```

```
    int i,row,col;

    fprintf(outFile,"P6\n %d %d 255\n",cols,rows);
    for (row = 0; row < rows; row++)
      for (col = 0; col < cols; col++)
        for (i = 0; i < 3; i++) {
          temp_char = (unsigned char) *image++;
          fwrite(&temp_char,1,1,outFile);
        }
  }

  unsigned char *upsample(unsigned char *tmp_image) {
    int i, j;
    unsigned char *new;

    new = (unsigned char *)malloc(sizeof(unsigned char)*pels*lines);
    for (i=0; i<lines/2; i++)
      for (j=0; j<pels/2; j++) {
        new[(2*i)*pels + 2*j] = tmp_image[i*pels/2 + j];
        new[(2*i)*pels + 2*j+1] = (tmp_image[i*pels/2 + j]+tmp_image[i*pels/2 + j+1])/2;
        new[(2*i+1)*pels + 2*j] = (tmp_image[i*pels/2 + j]+tmp_image[(i+1)*pels/2 + j])/2;
        new[(2*i+1)*pels + 2*j+1] = (new[(2*i+1)*pels + 2*j]+new[(2*i)*pels + 2*j+1])/2;
      }
    return(new);
  }

  unsigned char quant(float value) {

    int qval;

    qval = (value < 0.5 ?   (unsigned char) 0 :
            value > 254.5 ? (unsigned char) 255 :
                    (unsigned char) (int) (value + 0.5));
    return(qval);
  }

  unsigned char *convert(unsigned char *lum, unsigned char *cr, unsigned char *cb)
{
    int i, j, num;
    unsigned char *new;
    int yvalue, uvalue, vvalue;
    float rvalue, gvalue, bvalue;

    new = (unsigned char *)malloc(sizeof(unsigned char)*3*pels*lines);
    for (i = 0; i < lines; i++)
      for (j = 0; j < pels; j++) {
        num = j + i*pels;
        yvalue = lum[num];
        uvalue = cr[num] - 128;
        vvalue = cb[num] - 128;

        rvalue = yvalue + 1.4020*vvalue;
        gvalue = yvalue - 0.3441*uvalue - 0.7141*vvalue;
        bvalue = yvalue + 1.7720*uvalue;
```

```
      new[3*num] = quant(gvalue);
      new[3*num+1] = quant(rvalue);
      new[3*num+2] = quant(bvalue);
    }

  return(new);
}

main (int argc,char *argv[]) {
  int frm_num, i;
  unsigned char *new_image;
  char *filename1, *outfile;
  char imagename[256];
  unsigned char *lum, *cr, *cb, *tmp_image;

  argc--;
  argv++;
  if(argc != 5) {
    printf("Usage: <yuv2rgb imagename outfile frm_num pels lines>\n");
    exit(0);
  }
  filename1 = *argv++;
  outfile = *argv++;
  frm_num = atoi(*argv++);
  pels = atoi(*argv++);
  lines = atoi(*argv++);
  sprintf(imagename, "%s.%03d.Y", filename1, frm_num);
  if (!(inFile = fopen(imagename,"r"))) {
    printf("ERROR: Can't open %s for reading.\n", imagename);
    exit(0);
  }
  lum = loadimage(pels,lines);
  fclose(inFile);

  sprintf(imagename, "%s.%03d.U", filename1, frm_num);
  if (!(inFile = fopen(imagename,"r"))) {
    printf("ERROR: Can't open %s for reading.\n", imagename);
    exit(0);
  }
  tmp_image = loadimage(pels/2,lines/2);
  cr = upsample(tmp_image);
  fclose(inFile);

  sprintf(imagename, "%s.%03d.V", filename1, frm_num);
  if (!(inFile = fopen(imagename,"r"))) {
    printf("ERROR: Can't open %s for reading.\n", imagename);
    exit(0);
  }
  tmp_image = loadimage(pels/2,lines/2);
  cb = upsample(tmp_image);
  fclose(inFile);

  new_image = convert(lum,cr,cb);
```

```
  sprintf(imagename, "%s", outfile);
  outFile = fopen(imagename,"w");
  dumpimage(new_image, pels, lines);
  fclose(outFile);
  free(lum); free(cr); free(cb);
  free(tmp_image); free(new_image);
return;
}
```

This program produces an RGB image of size pels × lines, which needs to be converted into a 720 × 486 pixel image. This accomplished by the following C code:

```
#include <stdio.h>
#include <stdlib.h>

FILE *inFile;
FILE *outFile;
unsigned char *image;
int pels, lines;

unsigned char *loadimage()
{
  unsigned char temp_char;
  int i, row, col;
  unsigned char *image;
  unsigned char *res;

  fscanf(inFile, "P6 %d %d 255", &pels, &lines);
  image = (unsigned char *)malloc(sizeof(unsigned char)*3*pels*lines);
  res = image;
  fread(&temp_char,1,1,inFile);
  for (row = 0; row < lines; row++)
    for (col = 0; col < pels; col++)
      for (i = 0; i < 3; i++) {
        fread(&temp_char,1,1,inFile);
        if (feof(inFile)) {
          printf("ERROR: too few pixels\n");
          exit(0);
        }
        *res++ = temp_char;
      }
  fread(&temp_char,1,1,inFile);
  if (feof(inFile) == 0) {
      printf("ERROR: too many pixels\n");
      exit(0);
    }

  return(image);
}

void dumpimage(unsigned char *image, int cols, int rows) {
  unsigned char temp_char;
  int i,row,col;

  fprintf(outFile,"P6\n %d %d 255\n",cols,rows);
  fwrite(image, rows*cols*3, 1, outFile);
```

```
}

unsigned char *add_black(unsigned char *image) {
  int i, row, col;
  unsigned char *new;
  unsigned char *res;
  int tb, lr, lr2;

  new = (unsigned char *)malloc(sizeof(unsigned char)*3*720*486);
  res = new;
  tb = (486 - lines) / 2;
  lr = (720 - pels) / 2;
  for (row = 0; row < tb; row ++)
    for (col = 0; col < 720; col++)
      for (i = 0; i < 3; i++)
        *res++ = (unsigned char) 0;
  for (row = 0; row < lines; row++) {
    for (col = 0; col < lr; col++)
      for (i = 0; i < 3; i++)
        *res++ = (unsigned char) 0;
    for (col = 0; col < pels; col++)
      for (i = 0; i < 3; i++)
        *res++ = *image++;
    for (col = 0; col < lr; col++)
      for (i = 0; i < 3; i++)
        *res++ = (unsigned char) 0;
  }
  for (row = 0; row < tb; row++)
    for (col = 0; col < 720; col++)
      for (i=0; i<3; i++)
        *res++ = (unsigned char) 0;

  return(new);
}

main (int argc,char *argv[]) {
  unsigned char *new_image;
  char *filename1, *outfile;
  char imagename[256];

  argc--;
  argv++;
  if(argc != 4) {
    printf("Usage: <increase2_720x486 imagename outfile pels lines>\n");
    exit(0);
  }
  filename1 = *argv++;
  outfile = *argv++;
  pels = atoi(*argv++);
  lines = atoi(*argv++);
  sprintf(imagename, "%s.mv", filename1);
  if (!(inFile = fopen(imagename,"r"))) {
    printf("ERROR: Can't open %s for reading.\n", imagename);
    exit(0);
```

```
    }
    image = loadimage();
    fclose(inFile);

    new_image = add_black(image);
    sprintf(imagename, "%s.rgb", outfile);
    outFile = fopen(imagename,"w");
    dumpimage(new_image, 720, 486);
    fclose(outFile);
    free(image); free(new_image);
  return;
  }
```

Once the RGB image is the correct size (horse.720x486.rgb), it can be transferred to the Accom (location 500 in this example) using:

```
rcp horse.720x486.rgb accom:500.rgb
```

After all the frames of the sequence have been loaded to the Accom, the sequence can be viewed at any speed desired, it can be looped, played backwards, sections can be played separately, and the sequence can be recorded onto a VCR tape.

Bibliography

[1] A. Chandrakasan and R. Broderson, *Low Power Digital CMOS Design*, Kluwer Academic Publishers, Boston, 1995.

[2] J. Rabaey and M. Pedram, *editors*, *Low Power Design Methodologies*, Kluwer Academic Publishers, Boston, 1996, Chapter 1.

[3] A. Bellaouar and M. Elmasry, *Low Power Digital VLSI Design, Circuits and Systems*, Kluwer Academic Publishers, Boston, 1995, Chapter 1.

[4] A. Wu and K. Liu, "Algorithm-Based Low Power Transform Coding Architectures,", *Proc. ICASSP '95*, Vol. 5, 1995, pp. 3267-3270.

[5] J. Ludwig, S. Nawab, and A. Chandrakasan, "Low-Power Digital Filtering Using Approximate Processing," *IEEE Journal of Solid-State Circuits*, Vol. 31, No. 3, March 1996, pp. 395-400.

[6] T. Meng, B. Gordon, E. Tsern, and A. Hung, "Portable Video-on-Demand in Wireless Communication," *Proc. of the IEEE*, Vol. 83, No. 4, April 1995, pp. 659-680.

[7] D. Lidsky and J. Rabaey, "Low Power Design of Memory Intensive Functions Case Study: Vector Quantization," *VLSI Signal Processing, VII*, 1994, pp. 378-387.

[8] J. Biemond, L. Looijenga, D. Boekee, and R. Plompen, "A Pel-Recursive Wiener-Based Displacement Estimation Algorithm," *Signal Processing*, Vol. 13, 1987, pp. 399-412.

[9] J. Huang and R. Merserau, "Multi-Frame Pel-Recursive Motion Estimation for Video Image Interpolation," *Proc. ICIP '94*, Vol. 2, 1994, pp. 267-271.

[10] A. Netravali and B. Haskell, *Digital Pictures: Representation, Compression, and Standards*, Plenum Press, New York, 1995, Chapter 5.

[11] Motion Picture Experts Group, "ISO CD11172-2: Coding of moving pictures and associated audio for digital storage media at up to about 1-5 Mbit/s," November 1991.

[12] D. LeGall, "MPEG: A Video Compression Standard for Multimedia Applications," *Communications of the ACM*, Vol. 34, April 1991, pp. 46-58.

[13] H.-M. Hang, A. Puri, and D. Scilling, "Motion-compensated Transform Coding Based on Block Motion-tracking Algorithm," *IEEE International Conference on Communications '87*, Vol. 1, pp 136-140.

[14] Y. T. Tse and R. L. Baker, "Global Zoom/Pan Estimation and Compensation for Video Compression," *Proc. ICASSP '91*, 1991, pp. 2725-2728.

[15] V. Seferidis and M. Ghanbari, "Hierarchical Motion Estimation Using Texture Analysis," *IEE International Conference on Image Processing and its Applications*, April 1992, pp. 61-64.

[16] B. Liu and A. Zaccarin, "New Fast Algorithms for the Estimation of Block Motion Vectors," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 3, No. 2, April 1993, pp. 148-157.

[17] C.-H. Hsieh, P.-C. Lu, and J.-S. Shyn, "Motion Estimation Using Interblock Correlation," *1990 IEEE International Symposium on Circuits and Systems*, Vol. 2, May 1990, pp. 995-998.

[18] W. Li and E. Salari, "Successive Elimination Algorithm for Motion Estimation," *IEEE Transactions on Image Processing*, Vol. 4, No. 1, January 1995, pp. 105-107.

[19] S. Kim and C.-C. Kuo, "A Stochastic Approach for Motion Vector Estimation in Video Coding," *Neural and Stochastic Methods in Image and Signal Processing III, Proc. SPIE*, Vol. 2304, July 1994, pp. 111-122.

[20] A. Puri and R. Aravind, "Motion-Compensated Video Coding with Adaptive Perceptual Quantization," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 1, No. 4, December 1991, pp. 351-361.

[21] E. Salari and S. Lin, "Low-bit-rate Segmentation-based Image Sequence Coding," *Optical Engineering*, Vol. 34, No. 3, March 1995, pp. 829-832.

[22] S. C. Wells, "Efficient Motion Estimation for Interframe Video Coding Applications," *Electronics Letters*, Vol. 21, No. 7, March 1985, pp. 289-290.

6135-46