THE UNIVERSITY OF CALGARY

Real-Time Motion Picture Restoration

by

Remi J. Gurski

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER

ENGINEERING

CALGARY, ALBERTA

April, 1999

Abstract

Through age or misuse, motion picture films can develop damage in the form of dirt or scratches which detract from the quality of the film. Removal of these artifacts is a worthwhile process as it makes the films more visually attractive and extends the life of the material.

In this thesis, various methods for detecting and concealing the effects of film damage are described. Appropriate algorithms are selected for implementation of a system, based on a TMS320C80 video processor, which can remove the effects of film defects using digital processing. The restoration process operates in real-time at video frame rates (30 frames per second). Details of the software implementation of this system are presented along with results from processing damaged film material. The effects of damage are significantly reduced after processing.

Acknowledgements

I would first like to thank Dr. Bruton for his guidance and support throughout my graduate work, and especially for suggesting this field of research. It has been a challenging and rewarding experience. Norm Bartley has also been a great help in providing hardware and software solutions and also for tolerating a lot of good-natured abuse.

My lab-mates Chad Dreveny, James Gordy, and Joseph Provine have been a great source of ideas and lively conversation. I also thank Mark and Neil for reminding me of lunchtime so reliably each day.

Finally, I would like to thank NSERC for financial support over the past several years.

Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

ALU	Arithmetic Logic Unit
AMSE	Accumulated Mean Squared Error
AR	AutoRegressive Model
BBC	British Broadcasting Corporation
C80	Texas Instruments TMS320C80 processor
CBC	Canadian Broadcasting Corporation
DAC	Digital to Analog Converter
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
FIFO	First In, First Out memory buffer
FIR	Finite Impulse Response filter
GUI	Graphical User Interface
JOMBADI	JOint Model BAsed Detection and Interpolation
MAD	Mean Absolute Difference
MP	TMS320C80 Master Processor
MPEG	Motion Pictures Experts Group
MRF	Markov Random Field

MSE	Mean Squared Error
MX	Precision MX video processing card
NTSC	National Television System Committee
PAL	Phase Alternation Line
PC	Personal Computer
PCI	Peripheral Component Interconnect bus.
PP	TMS320C80 Parallel Processor
RAM	Random Access Memory
RGB	Red Green Blue colorspace
RISC	Reduced Instruction Set Computing
ROD	Rank Ordered Difference
SDI	Spike Detection Index
SDRAM	Synchronous Dynamic Random Access Memory
TC	TMS320C80 Transfer Controller
VC	TMS320C80 Video Controller
VGA	Video Graphics Array
VRAM	Video Random Access Memory
YUV	Luminance and Chrominance colorspace
I_n	Intensity components of frame n of an image sequence
$\hat{I}_n$	Motion-compensated frame I_n
$\tilde{I}_n$	Restored frame I_n
$\vec{r}$	A location (r_x, r_y) within an image (or frame)
$\vec{v}_{n-1}$	Motion vector between frames n and $n-1$

Chapter 1

Introduction

Motion picture restoration attempts to eliminate the effects of damage and age in motion picture films and return the film, as closely as possible, to its original undamaged state. This thesis investigates a number of techniques for restoring black and white archive films using digital processing, and outlines the design and implementation of a system which restores damaged film sequences in real-time.

Most classic movies and newsreels produced before 1950 are badly damaged [1]. Many of these have significant artistic and historic value. There is therefore an incentive for restoring these films and archiving them in digital form to prevent further degradation.

Observers find defects in motion pictures very distracting or annoying, since they often occur in random patterns and "don't belong" in the image sequence. Most of these defects must be removed before films are suitable for public viewing. In addition, compression methods (e.g. MPEG) do not work well on sequences with discontinuities such as scratches, and compression or image quality suffers [2]. Eliminating these defects improves coding performance.

With the introduction of digital television, broadcasters are finding that there

is excessive channel capacity available [3]. Using older film footage, such as movies or documentaries, is an inexpensive way to fill this excess capacity. Restoring this material and removing the effects of damage would improve the viewing quality. As mentioned earlier, removing defects also allows higher compression and therefore more efficient use of available digital television bandwidth.

A number of agencies, such as the BBC and CBC, have embarked on large restoration projects to preserve their rapidly deteriorating film archives. They have joined with various university and corporate researchers to develop improved methods of restoring film—some of these are the AURORA [3] and LIMELIGHT [4] projects. Their goal is quality of restoration first, and speed second. Most of the other motion picture restoration research currently being done is in developing a system which offers the best possible performance at any computational cost [3].

The methods examined in this thesis are not necessarily those which provide the best possible performance, but those which offer "acceptable" performance and are computationally efficient. Acceptable performance is defined as suitable for general-purpose viewing at television resolution, where most defects are removed but some effects of damage may remain. Any remaining damage is at a sufficiently low level as to be virtually imperceptible to the viewer. Creating a system which removes all film damage is extremely difficult, if not impossible, so this work concentrates on removing most damage as quickly as possible. In many cases, the quality of the real-time restoration algorithms is nearly identical to the most computationally intensive methods.

The advantages of a real-time system are the ability to connect it directly between a video source and a broadcast or recording medium, eliminating the need to preprocess the sequence. Real-time also represents the practical maximum speed

at which such a system needs to run; once real-time restoration has been achieved, then algorithms can be improved as more computational power becomes available.

Automatic film restoration in real-time has already been achieved for video processing systems from several companies such as Snell & Wilcox [5] and Digital Vision [6]. However, these are expensive special purpose units. The intent of the work in this thesis is to develop a system which requires the minimum amount of computation, using general-purpose hardware, in order to achieve adequate restoration quality.

Another criterion for this system is the ability to function with minimal operator intervention. Restoration should be robust to many types of sequences and damage, and the number of user parameters should be a minimum. The real-time restoration system developed in this thesis meets these goals.

The following sections describe typical types of film damage and an overview of the restoration process. A description of the test sequences used for this work and an outline of the thesis follows.

1.1 Film Damage

Over time, motion picture film degrades in quality due to a number of environmental factors. Dirt and mildew can accumulate on stored films, repeated use can scratch the film surface, and the chemicals in the film emulsion layer can deteriorate if not stored properly [7]. All of these effects are visible as the film is displayed and are distracting or annoying to viewers.

Figure 1.1 shows a movie frame with some simulated defects. Film damage may occur in the following forms [3]:

Figure 1.1: Some examples of film damage demonstrated on a frame from the original WESTERN sequence.

- Dirt, mold, and other foreign objects adhere to the film surface and appear as dark lines or "blotches". Scratches or other light marks on the film negative will appear dark when the film is printed.

- Scratches on the film create light lines or spots (sometimes called "sparkle"). Older nitrate stock based film deteriorates over time and its emulsion layer flakes off, also leaving bright areas. Dirt or dark marks on the negative will appear light when the film is printed.

- Dirt particles in the film projector can cause long vertical scratches (line scratches) as the film passes through the transport mechanism. They may appear light or dark, depending on whether the negative or the printed film is damaged.

- Film grain noise is created by slight imperfections in the chemical film layer. This may appear as impulsive "salt and pepper" noise, where the defects are usually only a single pixel in size.

- Miscellaneous other defects may be the result of water damage or excessive heat.

Examples of these forms of damage are shown in the sample image.

It is important to distinguish between actual dirt which has adhered to the film surface, and "printed" dirt, which was present when the film was created and therefore part of the image. Both appear the same when the film is viewed, but while actual dirt may be removed by cleaning, printed dirt cannot. Also, the removal of scratches depends on whether they are on the base "support" side of the film, or on the emulsion side, which bears the chemical coating and actual image [8]. Base side

scratches simply obscure the image when light is shone through, while emulsion side damage actually eliminates image information.

In order to remove or conceal its effects, the characteristics of film damage must be known. With the exception of line scratches, defective regions will appear in random locations in each frame, i.e. it is rare for a damaged pixel in frame n to also be damaged in frame $n + 1$. The reason for this is the purely random nature of dirt and scratches: if it is equally likely to occur at any location in a frame, the probability is low that it will occupy the same location in adjacent frames.

Vertical line scratches are a special case, because they will appear in almost the same location over many successive frames. The same methods used to conceal random film damage cannot be used. Instead, the fact that they are relatively narrow, occupy the full height of the frame, and occur in approximately the same place in several frames can be used to detect line scratches [9].

A number of other artifacts can occur as a result of film degradation or misuse. These include bleaching of colors, discoloration of both black and white and color films, wearing of the film sprocket holes causing "jitter", shrinkage, shake, and "flicker" caused by random differences in frame brightness [2]. Conversion of film to video can also cause problems due to synchronization differences between video lines, resulting in "line jitter" [10].

1.2 Restoration Techniques

This thesis is concerned primarily with the detection and concealment of random film damage, such as dirt and scratches. Techniques which are computationally efficient and can be performed in real-time are specifically considered. These methods operate exclusively on black and white (greyscale) image sequences, not because the

techniques are limited to greyscale (see Section 7.2), but for the lower processing requirements. All input sequences are assumed to have been previously converted to a video signal, and the resulting restored signal is also video. A high-resolution film scanner could also be used as an input device, although at a higher computational cost because of the greater resolution.

There has been some success in removing flicker and jitter in motion picture films [2], as well as color restoration [3]. Removal of these artifacts is beyond the scope of this thesis, so these methods are not investigated. Line scratches have different characteristics than dirt and random scratches and must be treated separately, so the removal of line scratches is also not considered in this work.

1.2.1 Traditional Film Restoration

The most straightforward method of removing dirt from film is to carefully wash deposits from the surface by hand or with an ultrasonic cleaning technique [8]. This obviously will not work for scratches or printed dirt. One method of detecting physical dirt uses infrared light projected through the film [11]. Regardless of its transparency to visible light, the film will pass infrared except at those locations which contain dirt particles. These locations must then be restored by cleaning or by some interpolation technique, such as those described in Chapter 4.

Small base side scratches can be removed using a "wetgate" method [8], which coats the film with a chemical having a refractive index close to that of the film material. Light projected through the film and chemical layer will not refract at scratch locations and they will therefore not appear. Some superficial damage to the emulsion layer can also be removed by the wetgate method.

If the film is badly damaged, the only method of restoring it may be to use

an airbrush and retouch each frame by hand, very slowly and at great expense.

1.2.2 Digital Restoration

As digital techniques are used more in the motion picture industry for archiving and special effects, techniques for using digital processing to restore damaged films have been proposed. One such method is an extension of manual restoration, where an image processing workstation and software are used instead of an airbrush [12]. While this may be more convenient, the cost in time and money is still large.

Several researchers have devised methods for automatically restoring damaged films with minimal operator supervision [13, 3, 4, 14]. Such algorithms detect damage and conceal it using image processing. This method is certainly faster and less expensive than manual restoration. A number of techniques using heuristics, autoregressive models, and other algorithms have been successfully applied to this problem and are described in detail in later chapters.

All of the digital restoration techniques use the same basic method for processing [15]. At film rates (24 frames per second), the difference in information between two adjacent frames will be small. Because of the temporally local nature of film damage (at any given pixel location), information from the surrounding frames can be used to detect damage and reconstruct damaged regions. However, moving objects will appear at different locations in the two frames. A motion estimation and compensation algorithm detects motion and adjusts for the effects of movement between frames. Processing can then be carried out assuming that there is no motion between frames.

Typically, three frames at a time are used for restoration: the current frame to be processed, and the previous and next frames in the sequence, as shown in

Figure 1.2. The previous and next frames are compensated for motion and used,

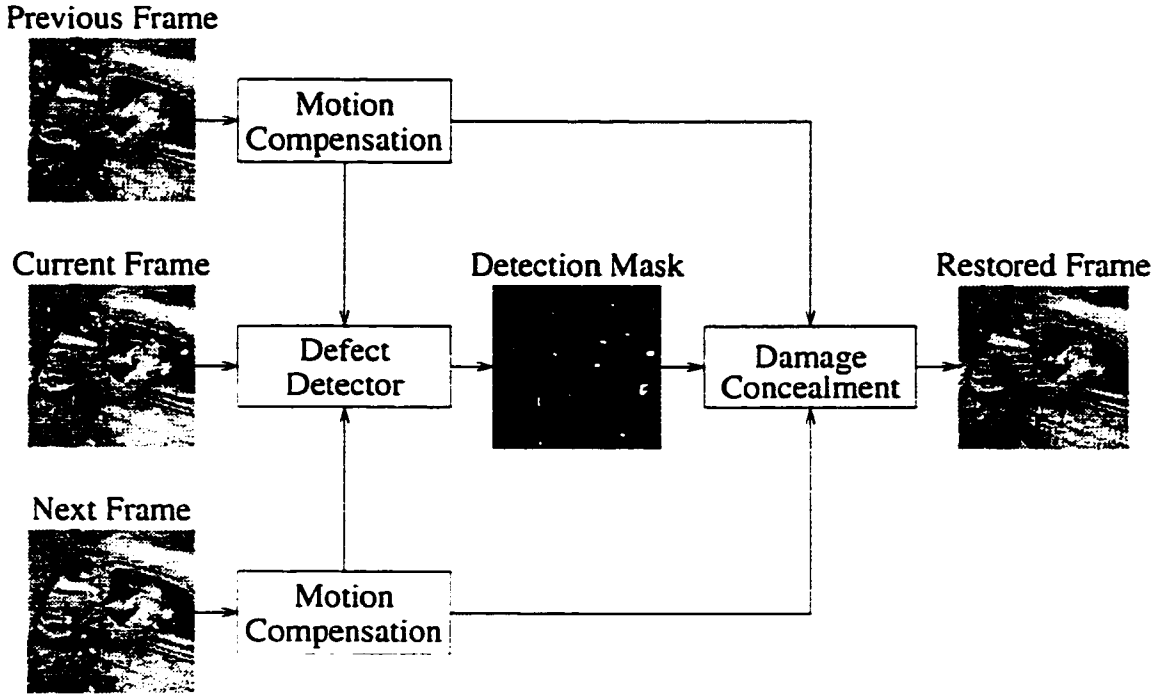


Figure 1.2: Steps to restore a single frame from an image sequence.

along with the current frame, in a detection algorithm that attempts to find which locations (if any) in the frame are damaged. The result of this step is a detection mask—a bitmap indicating the damage locations. A concealment algorithm uses information from the surrounding frames (and possibly the current frame) to cover only the locations marked in the detection mask. The result is a restored frame which closely resembles the original undamaged frame.

A number of specialized hardware systems applied to the restoration problem have been developed. These include a parallel array of digital signal processors [16], the Philips-BTS MNR-11 digital noise reducer [17], and video processing workstations from Snell & Wilcox and Digital Vision. Some of these systems are able to perform film restoration in real-time (24 frames per second).

1.3 Film to Video Transfer

The motion picture restoration system described in this thesis operates on a North American standard NTSC video signal input. It is assumed that a film sequence has already been converted to a video signal and either input directly to the restoration device or recorded on videotape for later processing. The reason for developing the system for video processing is because a digital film scanner is expensive and not readily available, so previously digitized sequences or videotape are convenient sources of film material. Also, the lower resolution of video allows the use of less powerful processing hardware, while the algorithms can later be extended to higher resolutions.

Film is converted to a video signal through the use of a "telecine", which internally projects the film and scans it into video format [18]. The telecine also performs the necessary operations to convert the film rate from 24 frames per second to an appropriate video frame rate. An illusion of more frames per second is therefore created without altering the overall speed of motion in the film.

NTSC video effectively displays 30 frames per second, but each frame is divided into an odd and even field. Fields are therefore displayed at 60 per second[1], and are interlaced: the odd field contains the odd lines of a frame, while the even field contains the even lines. Film frames are usually converted to video format using a "3:2 pulldown" process [19], where film frames are repeated as necessary to convert the frame rate to an average of 30 frames per second. Frames from the film are alternately repeated for 2 or 3 fields: the result is two film frames converted to five NTSC fields, for the required conversion ratio of 2.5 (60 fields per second divided by 24 film frames per second). For the European PAL video standard,

[1] The exact rate is 59.94 fields per second [19].

which operates at 25 frames per second, film frames are usually directly converted to video frames, as the difference in frame rates is not noticeable to viewers.

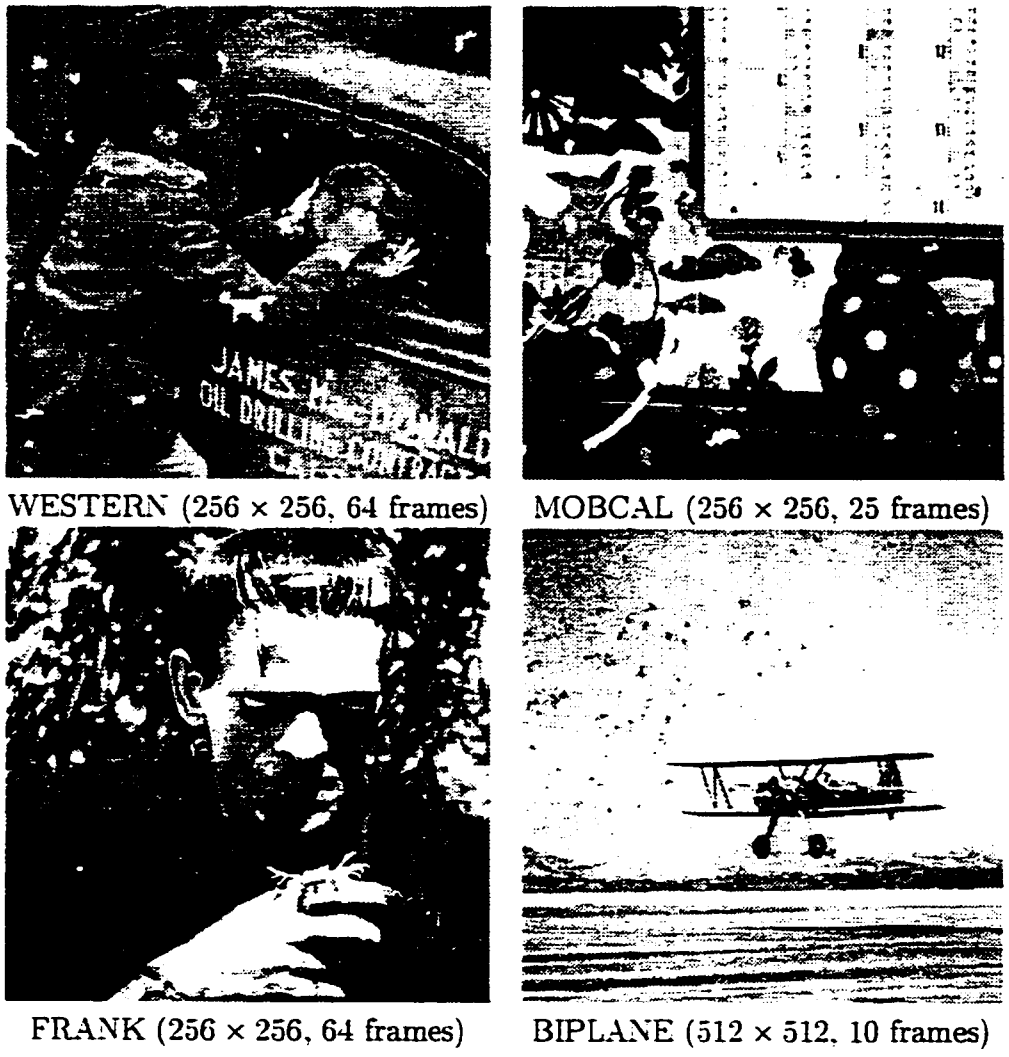
For film converted to NTSC video, the fields may contain information from the same film frame or from adjacent frames, depending on the position in the sequence. This can cause a problem when processing video frames—the information in the odd lines may not correspond with the information in the even lines.

The resolution of modern 35mm motion picture film has been estimated to be at least 4096 × 2988 pixels per frame, with 10 to 12 bits per pixel per color, at 24 frames per second [4]. In contrast, NTSC video resolution is approximately 720 × 486 pixels at 8 bits per pixel per color, at 30 frames per second [19]. The spatial information alone is a factor of 35 greater for film. Algorithms described in this thesis for motion picture restoration of video, however, are scalable to full film resolution assuming adequate computing power is available.

1.4 Test Sequences

A number of test sequences are used in this thesis to evaluate the performance of various restoration algorithms. Some contain synthetic damage and others are digitized from actual damaged films. Example frames from all the sequences are shown in Figure 1.3. All sequences were obtained from Anil Kokaram and Cambridge University [3].

The WESTERN sequence was digitized from an undamaged motion picture film, and consists of 64 frames at 256 × 256 pixels, 8-bit greyscale. The actor in the scene lifts a sack into the truck, then adjusts his jacket. It contains a fair amount of stationary background, as well as some fast and complex movement. MOBCAL (Mobile-Calendar) is a standard video test sequence which was obtained from a



WESTERN (256 × 256, 64 frames)

MOBCAL (256 × 256, 25 frames)

FRANK (256 × 256, 64 frames)

BIPLANE (512 × 512, 10 frames)

Figure 1.3: Sample frames from test sequences.

camera, not a film, and consists of 25 frames at 256 × 256 8-bit greyscale. The black and white balls spin and intersect, while the calendar moves up and down and the train pushes the spotted ball. A wide range of movement is seen in all parts of the image, with fairly detailed objects.

Both WESTERN and MOBCAL had artificial damage added, in the form of blotches of random greyscale value at random locations [20]. The original undamaged sequence is available and the artificial damage locations are known, so the performance of the restoration algorithms can be quantitatively evaluated against the original. These sequences were used extensively in Chapter 5 to select the best algorithms for motion estimation, defect detection, and damage concealment.

The FRANK (256 × 256, 64 frames greyscale) and BIPLANE (512 × 512, 10 frames greyscale) sequences were digitized from actual damaged films. The FRANK sequence consists of a slow camera pan upwards as the Frankenstein monster lifts a flower and the leaves in the background blow in the wind. Most of the sequence is undamaged, with a notable exception being the spot on the forehead shown in the figure. In BIPLANE, the camera follows a landing airplane while the background scrolls because of the camera pan. The damage is significant, especially in the frame shown in Figure 1.3.

In addition to these sequences, a variety of black-and-white movie and news documentary footage was recorded from television and used for testing. Some results from testing on these sequences are shown in Section 6.3.

1.5 Outline of the Thesis

The first chapter of this thesis has presented the background and motivation behind motion picture restoration, as well as stating the intent of this research: to develop

an automatic, real-time restoration tool which uses general-purpose hardware (and is therefore inexpensive). The focus application is video to video transfer, where the input and output formats are video signals, but the algorithms are extensible to higher (film) resolutions. Typical types of film damage and an overview of methods previously used to remove these artifacts were also given.

In Chapter 2, the motion estimation and compensation process is described in detail. Motion estimation techniques can be basically divided into gradient, frequency domain, and block matching methods, with block matching being the most appropriate for real-time processing. Several block matching metrics and search algorithms are presented and compared on the basis of accuracy and computational cost. Issues related to motion estimation, such as image scaling and acceptance thresholds, are described.

The discussion in Chapter 3 concerns possible techniques for detecting damaged locations in film frames. A number of algorithms are described to detect "random" damage (scratches and dirt). These techniques have widely varying performance and computational cost. Finally, a discussion on possible ways to eliminate vertical line scratches is given.

In Chapter 4, the final concealment step of the restoration process is explored. Various averaging or median filtering methods are described, as well as more complex (and computationally expensive) algorithms using Markov random fields and autoregressive models.

In Chapter 5, the design of the restoration software is explained. First, an overview of the implementation platform is given. Motion estimation, detection, and concealment methods are then compared with several variations and evaluated on the basis of accuracy and computational cost. The best algorithms for each are

selected as candidates for a real-time implementation.

The software implementation and structure of the real-time restoration system, its capabilities, and its user interface are all described in Chapter 6. Restoration performance is demonstrated on real damaged film sequences. Execution speed of the final implementation with various processing options is presented.

Finally, the discussion in Chapter 7 summarizes the performance of the system, its problems, and possible improvements.

Chapter 2

Motion Estimation and Compensation

It is possible to use adjacent frames in an image sequence to detect and conceal damage within a given frame. These surrounding frames contain almost the same information as the center frame, assuming they are not on the boundary of a scene change. However, if there is motion in the scene (or if the camera moves), there will be slight differences in the locations of objects within the frames. Motion estimation attempts to find the magnitude and direction of the motion, while motion compensation reduces the effect of motion between the frames. Accurate motion estimation and compensation are necessary for detection and concealment of damage.

Motion estimation attempts to find a displacement vector between pixels in two successive frames, such that [20]

$$I_n(\vec{r}) = I_{n-1}(\vec{r} + \vec{v}_{n-1}(\vec{r})) + \epsilon(\vec{r}) \tag{2.1}$$

where $\vec{r}$ is a two-dimensional coordinate with components (r_x, r_y), $I_n(\vec{r})$ is the pixel intensity at location $\vec{r}$ in frame n, and $\vec{v}_{n-1}(\vec{r})$ is the motion vector between frames

n and $n-1$ at location $\vec{r}$. The motion estimation algorithm attempts to find the best value for $\vec{v}_{n-1}(\vec{r})$ for all locations $\vec{r}$ to minimize the value of the error term $\epsilon(\vec{r})$.

Once motion estimates have been obtained for all pixels in the frame, inter-frame processing can use these offsets when selecting pixels in order to minimize the effects of motion between frames. To compare pixels at the same effective (i.e. motion invariant) location in the current and reference frames, the vector $\vec{v}_{n-1}(\vec{r})$ must be added to each pixel location when accessing pixels from the reference frame. In order to simplify this process, the reference frame can be preprocessed to compensate for the effects of motion:

$$\hat{I}_{n-1}(\vec{r}) = I_{n-1}(\vec{r} + \vec{v}_{n-1}(\vec{r})) \tag{2.2}$$

The motion compensated frame $\hat{I}_{n-1}(\vec{r})$ can then be used instead of the reference frame $I_{n-1}(\vec{r})$, leading to the simplified motion relation:

$$I_n(\vec{r}) = \hat{I}_{n-1}(\vec{r}) + \epsilon(\vec{r}) \tag{2.3}$$

In the restoration process, motion estimation is performed both forward and backward, so $\hat{I}_{n+1}(\vec{r})$ is also calculated and used in a similar way. Motion compensation can be performed explicitly immediately after motion estimation or, alternatively, vector offsets can be added during inter-frame processing, with identical results. Throughout this thesis, motion compensated reference frames $\hat{I}_{n-1}$ and $\hat{I}_{n+1}$ are assumed in order to simplify notation.

This motion model assumes movement is purely translational and parallel to the image plane [19]. It also assumes that the brightness levels of both images are the same, and that objects are not occluded (i.e. move behind another object) or unoccluded during their movements. These are not always realistic assumptions, but nevertheless the motion estimation process provides an acceptable approximation for

restoration [21].

It is important to note the difference between actual motion and optical flow [21]. A film or video camera records the positions of objects at specified moments of time and translates this information into a two-dimensional image. The information in an image sequence, therefore, does not contain any explicit information about the movement of objects in three-dimensional space. Instead, only the time-varying intensity of the images as a function of time—the optical flow—is available. Motion estimation algorithms using Equation 2.1 estimate only optical flow, and therefore may not determine the "true" motion in a scene. For example, when a large object of constant intensity moves across the camera's field of view, the optical flow near the center of the object is constant because the intensity of those pixels does not change with time. In this case, no motion will be detected. Most objects, however, have intensity variations across their surface and in contrast with background objects. In these cases, optical flow is an accurate measure of actual motion (within the assumptions of the motion model) for the purposes of restoration.

For color sequences, motion estimation is usually performed only on the luminance components, in exactly the same way as previously described [19]. More complex methods have also been proposed which use both luminance and chrominance components for greater accuracy [22]. The work in this thesis considers only luminance components for speed and simplicity, as most damaged film sequences were not filmed in color.

2.1 Estimation Techniques

Many methods have been developed for estimating motion in image sequences. The following sections describe several categories of estimation techniques and their suitability for a real-time restoration system. Gradient methods use a Taylor series expansion of the frame difference equation along with a smoothness requirement to generate a motion estimate for each pixel in a frame. Frequency domain techniques rely on the frequency characteristics of motion to calculate an appropriate displacement. Finally, the block matching technique compares blocks of pixels at various offsets to directly determine the displacement with the most similarity between frames.

2.1.1 Gradient Methods

Gradient based methods for motion estimation calculate a motion displacement for each pixel in a frame based on the assumption that luminance is constant along motion trajectories [21]. Under this assumption, the term $\epsilon(\vec{r})$ in Equation 2.1 is zero. Expanding the remaining term $I_{n-1}(\vec{r}+\vec{v}_{n-1}(\vec{r}))$ into a first-order Taylor series gives the following expression [20]:

$$I_n(\vec{r}) = I_{n-1}(\vec{r}) + \vec{v}_{n-1}(\vec{r}) \cdot \vec{\nabla} I_{n-1}(\vec{r}) \tag{2.4}$$

where $\vec{\nabla}$ is the two-dimensional gradient function. In order to calculate both the x and y components of the motion estimate $\vec{v}_{n-1}(\vec{r})$, another equation must be used. Typically, this is a smoothness constraint which assumes that pixels within a small $N \times N$ block have similar motion vectors. One possible formulation is [3]:

$$\mathbf{z_0} = \mathbf{Gv} \tag{2.5}$$

where $\mathbf{z_0}$ is a N^2 entry column vector of frame difference values $(I_n(\vec{r}) - I_{n-1}(\vec{r}))$ for all pixel locations $\vec{r}$ within an $N \times N$ block and $\mathbf{G}$ is a $2 \times N^2$ array of x and y gradients from frame I_{n-1}. Solving the resulting equations for $\mathbf{v}$ yields motion estimates for each pixel.

The gradient motion estimation technique is accurate only for small motion displacements because the Taylor series approximation is valid only over small distances [7]. A number of ways to expand the search region have been proposed, including pixel-recursive and hierarchical techniques [3], but in general the potential search area is smaller than for frequency or block-based methods. It is also fairly susceptible to noise during the gradient calculations.

Computational requirements are low for gradient motion estimation, as compared to other methods in terms of operations per pixel. However, the density of the vector field presents an implementation problem: as each pixel within a block potentially has a different displacement, data access will not be predictable or linear within the frame buffer. The overhead of performing motion compensation (irregular memory transfers), therefore, will likely offset the lower computational cost.

While gradient estimation is attractive for its computational efficiency, it is unable to estimate the large motion displacements present in many films. Its susceptibility to noise is also a major factor when processing damaged motion pictures. Some experimental results between gradient and block matching algorithms show significantly better results when using block matching [21, 3]. For all of these reasons, the gradient method is judged to be unsuitable for a restoration application. More appropriate estimation techniques are discussed in the following sections.

2.1.2 Frequency Domain Methods

Frequency domain motion estimation uses the frequency characteristics of motion in order to calculate displacement vectors. One popular method is phase correlation [23], which uses the time (or space) shifting property of the Fourier transform [24]:

$$x(n-k) \stackrel{\mathcal{F}}{\longleftrightarrow} e^{-j\omega k}X(\omega) \tag{2.6}$$

Applying this property to image sequences, motion resulting in a shift of v_x and v_y in the x and y directions, respectively, will cause a phase shift in the frequency domain relative to the phase for a stationary sequence. By detecting the magnitude of the phase shift in each dimension, the pixel offset can be determined.

To determine motion offsets, blocks from two successive frames are chosen and their Fourier transforms are calculated. The magnitude of both spectrums are normalized to reduce the effects of brightness changes between frames. The phase from one image is subtracted from the phase of the other for each frequency, then the inverse Fourier transform is calculated on the difference signal. Peaks will appear in the resulting image at locations which correspond to possible motion displacements.

The interpretation of the phase peaks is implementation dependent. In the simplest case, the largest peak is chosen as the dominant motion offset and all pixels within the block are assigned that estimate. More sophisticated methods [23] will translate the image block by each possible displacement in turn, calculate the error between the two frame blocks, and assign the estimate only to those regions with low error. In this way, complex moving regions can be accurately tracked.

Motion can be very accurately determined through phase correlation. The computational overhead is fairly high because the Fourier transform must be calculated for each block and its surrounding search window, then phase difference and

inverse Fourier transform are calculated. More sophisticated methods require even more calculation to determine the best estimate for all locations within the block. Compared with the block matching estimation method, phase correlation requires more computation for small search windows but eventually is more efficient for larger windows. For one implementation, with a maximum motion vector displacement of 8 pixels and blocks of size 8×8, a simple phase correlation method requires half the operations as an exhaustive block search [23], although the operations are less complex for block matching.

Phase correlation is promising for use in a restoration system, but its computational complexity is too high for real-time estimation using the current system (see Section 5.1). Reduced search methods make the block matching method more efficient even for very large search windows. Phase correlation is therefore considered to be impractical for the real-time restoration system.

2.1.3 Block Matching

In the block matching motion estimation technique [19], the current frame is divided into equal-sized blocks of size $L \times L$ pixels. All pixels within the block are assumed to belong to the same (possibly moving) object, and therefore a single motion vector is calculated which applies to all L^2 pixels[1]. Each block from the current frame is compared with a block of equal size in the reference frame, at several locations within the valid search window W. The offset of the "best" match, determined by the minimum of some comparison metric $d()$, is chosen to be the optimal motion vector for the current block. The search process for a single $L \times L$ block is therefore [21]:

$$\vec{v}_{n-1}(\vec{r}) = \vec{p} : \min d(I_n(\vec{r}), I_{n-1}(\vec{r}+\vec{p})) \qquad |p_x| \leq W, |p_y| \leq W \qquad (2.7)$$

[1]Per-pixel estimates can be made through bilinear interpolation with surrounding blocks [25].

where I_n and I_{n-1} are the current and previous frames, respectively, and $\vec{r}$ is the position of the block within the frame.

Block comparisons in the reference frame are typically performed within a square of size $L + 2W$, with the offset $(0,0)$ corresponding to the location of the block in the current frame, as shown in Figure 2.1. The total number of possible

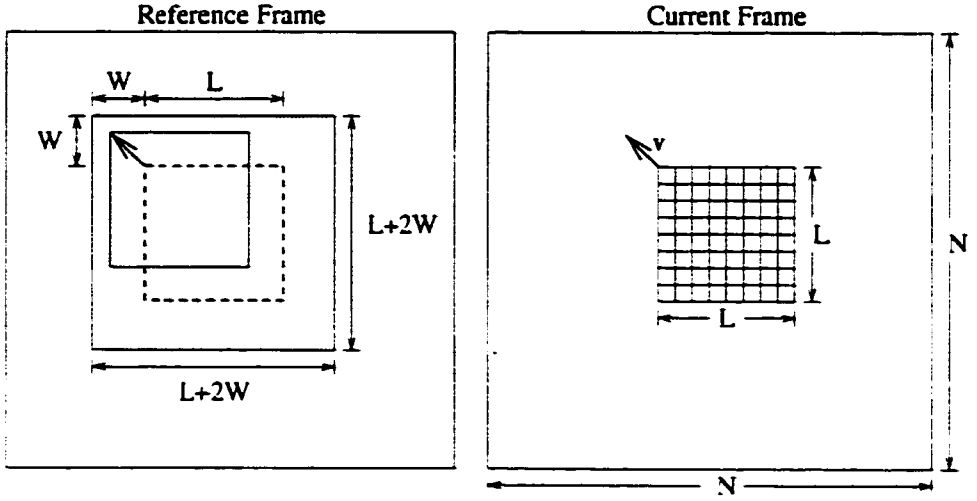


Figure 2.1: The block search technique.

vectors for a single block is therefore $(2W + 1)^2$, with a range from $(-W, -W)$ to (W, W). Depending on the search method used (see Section 2.3), not all offsets may be searched. It is also possible to determine offsets with sub-pixel accuracy by using bilinear interpolation between pixel locations in the reference frame [25].

The dimension L of the motion blocks has to be selected as a tradeoff between vector granularity and noise immunity [25]. Small values of L will divide the image into many small blocks which should offer a better approximation of the shapes of moving objects. However, the effects of impulsive noise can cause incorrect motion estimates, because each single pixel has a relatively higher significance in the block comparison. Larger block sizes are more immune to noise but provide poor approximations of moving regions. Values for L of 8 or 16 are most common in

practice [25].

The block matching technique provides good results for motion estimation, even at fairly large offsets. It suffers in some circumstances due to the assumption that motion is purely translational between frames. Another problem is that objects rarely have edges which correspond exactly with block boundaries, so some blocks will contain a portion of a moving object and some stationary background. The motion offset will therefore vary between no motion (if the background is more significant) to the correct motion estimate, or a random vector in-between. A similar problem occurs when regions are uncovered or objects move in and out of the frame.

Compared with gradient or frequency domain motion estimation, block matching is neither the fastest nor the most accurate for estimating "real" motion. However, comparisons based on frame difference and entropy metrics have shown that block matching provides performance equal to or better than the gradient estimation method in image coding applications [21]. Block matching also has the advantage of requiring only simple operations (addition/subtraction and absolute value) compared with other estimation methods. For these reasons, block-based motion estimation has been chosen to be the most suitable for a real-time restoration system. The following sections deal exclusively with block matching estimation algorithms.

2.2 Block Comparison Metrics

In order to determine the best vector offset between two blocks, they have to be compared and a measure of their similarity at some given offset must be obtained. A comparison metric is a mathematical norm between two blocks which quantitatively represents their similarity.

Mean square error (MSE) is a common comparison metric [19]. Assume that

blocks are of size $L \times L$. The current frame is I_n and the reference frame is I_{n-1}. The MSE is:

$$MSE \equiv \frac{1}{L^2} \sum_{\vec{r} \in \mathbf{L^2}} (I_n(\vec{r}) - I_{n-1}(\vec{r} + \vec{v}_{n-1}))^2 \tag{2.8}$$

where $\mathbf{L^2}$ is all locations within the block and $\vec{v}_{n-1}$ is the motion vector for that block.

The calculation of the MSE requires three operations per pixel—a subtraction, multiplication, and an addition[2]. With L^2 pixels in a block, $3L^2$ operations in total are required to perform one block comparison. This assumes a multiplication can be performed as quickly as an addition or subtraction, which may not be the case in all hardware. For this reason, the mean absolute difference (MAD) [25] is sometimes used instead of the MSE:

$$MAD \equiv \frac{1}{L^2} \sum_{\vec{r} \in \mathbf{L^2}} |I_n(\vec{r}) - I_{n-1}(\vec{r} + \vec{v}_{n-1})| \tag{2.9}$$

The MAD also requires $3L^2$ operations, but uses absolute value instead of multiplication and therefore may be faster on some hardware.

Both the MSE and MAD methods require on the order of L^2 calculations per block. One technique which requires a reduced number of operations is the Radon projection [13], which is shown in Figure 2.2. Instead of calculating the difference between each pair of pixels in the current and reference block, the projection along each row (r_i) and column (c_i) is first calculated for each block:

$$r_{n,i} = \sum_{j=0}^{L-1} I_n(r_x + j, r_y + i) \qquad\qquad c_{n,i} = \sum_{j=0}^{L-1} I_n(r_x + i, r_y + j)$$

where the upper-left corner of the block is at location $\vec{r}$ in frame I_n. The difference between the projections is then used to calculate the block difference:

$$E_P = \sum_{i=0}^{L-1} ((r_{n,i} - r_{n-1,i})^2 + (c_{n,i} - c_{n-1,i})^2) \tag{2.10}$$

[2]The division by L^2 can be ignored if all blocks are a constant size.

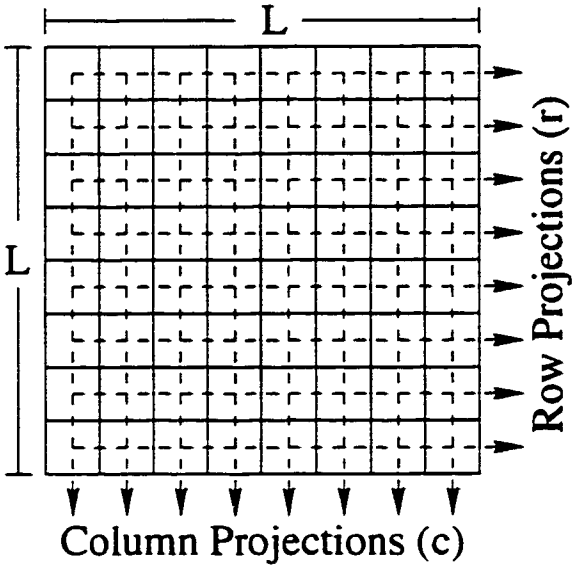


Figure 2.2: Calculation of block projections.

The projection method requires $2L^2$ additions to calculate the projections for each block and $6L$ operations for the comparisons. The projections for the current block need only be calculated once for all offsets. For the reference block, all projections need not be recalculated for each location: only the new column or row (L operations), and the old rows (or columns) need to be adjusted by subtracting the old end pixel and adding the new end pixel ($2L$ operations). The projection method therefore requires $4L^2 + 6L$ operations for the first location, and $9L$ operations for each following location.

The projection method has a higher computational cost for the first search location but much lower computation for the following locations. For a given value of L, there is a minimum search window size before it requires fewer operations than the MSE or MAD methods. Its accuracy may also be lower than for the MSE or MAD case because it only compares pixel sums instead of individual pixels [13].

2.3 Block Search Algorithms

A variety of techniques can be used to determine the best motion vector within the search window. The simplest form, exhaustively comparing the current block with every possible location in the search window, is computationally intensive. Other methods are available which rely on searching only a subset of locations to converge on the optimal (or near-optimal) displacement. The following sections describe several techniques for searching for the best match. This is not a complete list of possible search algorithms; only the best or significantly different methods are described here. Some other algorithms not mentioned are field or block decimation [26], adaptive segmentation [21], or one-dimensional (PHODS) search [25, 19].

2.3.1 Exhaustive Search

The exhaustive motion search algorithm [19] compares the block in the current frame with all possible locations within the search window in the reference frame. Valid motion vectors are therefore any values within the range of $(-W, -W)$ to (W, W). This technique is guaranteed to find the "best" motion vector, as defined by the comparison metric and within the assumptions of the motion model [25]. Its drawback is the number of operations required to compute the motion vectors. With a maximum search window offset W, the total number of locations C to search per block is[3]:

$$C = (2W + 1)^2 \tag{2.11}$$

A block comparison calculation must be performed at each location to determine the best motion vector.

[3] Assuming all locations within the search window lie within the reference frame—this assumption is used for the analysis of all motion search algorithms.

Exhaustive search is useful when the accuracy of motion estimates is more important than computational cost. However, it is possible to calculate motion vectors very close or equal to the optimal value by considering only a small subset of all possible search locations. These faster techniques achieve good results at a small fraction of the computation required for exhaustive search. Some of these methods are described in the following sections.

2.3.2 Overlapped Search

Block matching using overlapped blocks [16] is a method to improve the granularity of motion estimates without sacrificing noise immunity. As described in Section 2.1.3, small block sizes better approximate the outlines of moving objects but are susceptible to impulsive noise, which can result in incorrect estimates. Overlapping block search uses larger blocks for the comparison operation and assigns the motion vector to a smaller block of pixels. Unlike the other search methods described in later sections, the overlap search technique requires more computation than exhaustive search but can provide better results.

Figure 2.3 illustrates this technique. The current frame is divided into blocks of $L \times L$ pixels, as before. A larger block of size $M \times M$ surrounds the smaller block. An exhaustive block search is performed on these $M \times M$ blocks in the current and reference frames. When the best match is obtained, only the pixels within the smaller $L \times L$ block use this estimate. The larger blocks overlap some of the same area of the current frame in adjacent blocks, thus the name.

With $M = L$, the algorithm is identical to exhaustive search. By setting $L < M$, the motion estimation process is more immune to noise because of the large $M \times M$ search blocks. At the same time, the granularity of the motion vectors is

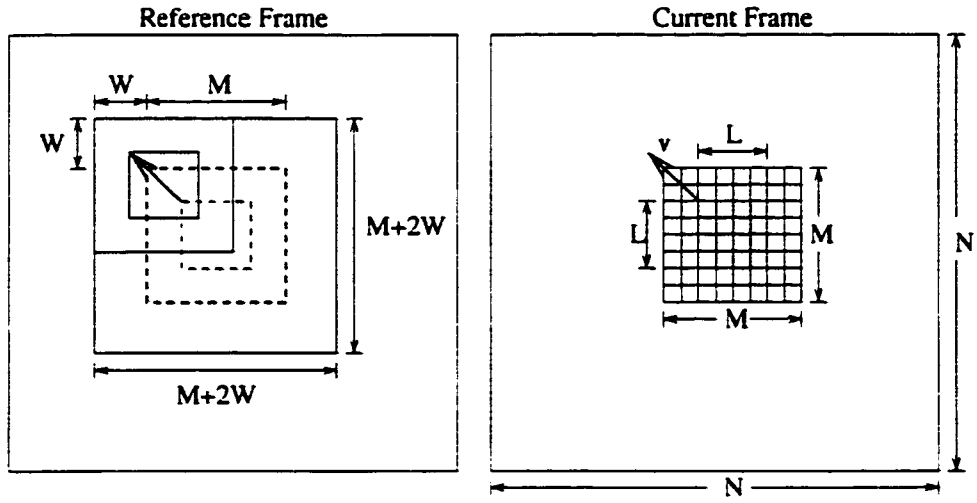


Figure 2.3: Overlapped block search method.

increased because vectors are only assigned to the smaller $L \times L$ block.

The number of vector offsets searched for each block is the same as for exhaustive, $(2W + 1)^2$. However, the comparison metric requires more operations to perform, since there are M^2 pixels involved in the block comparison for each block of L^2 pixels in the frame. The number of operations per $L \times L$ block can be adjusted to reflect this fact by multiplying by a factor[4] of $\frac{M^2}{L^2}$. The total number of locations C searched per $L \times L$ block is therefore:

$$C = \frac{M^2}{L^2}(2W + 1)^2 \tag{2.12}$$

The computational cost is very high but the results can be significantly better compared to the exhaustive technique, as illustrated by the experimental results in Section 5.2.1.

[4]This is accurate for the MSE and MAD methods; Radon projection requires a more complex scaling factor which is approximately equal to $\frac{M}{L}$.

2.3.3 Logarithmic Search

In order to decrease the number of locations checked within the search window, heuristic approaches can be used which only check a subset of all possible locations to approximate the optimal vector. One of these methods is the logarithmic search [25], also called the n-step search [19].

Figure 2.4 shows the search window and the offsets considered for comparison. In the first step (indicated by "1" in the figure), nine locations surrounding the zero

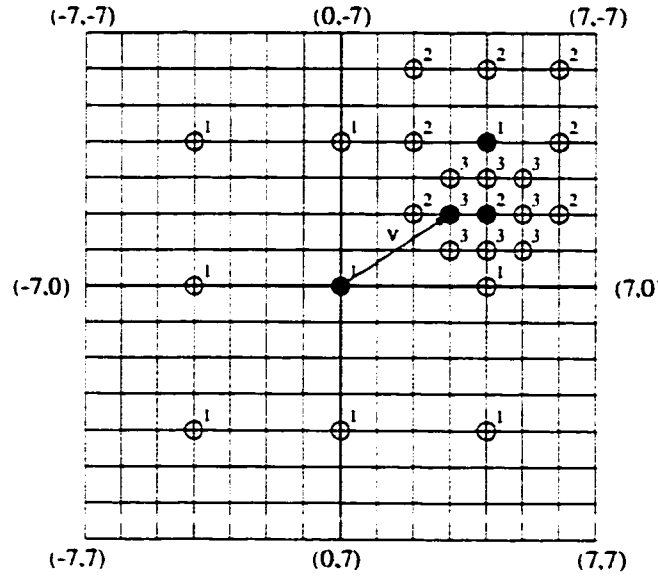


Figure 2.4: Logarithmic search method. Best matches are shown as dark circles.

offset are considered, at a distance of 2^{n-1} from the center, where n is the number of steps used in the algorithm. After the best match of these nine locations has been determined, the second step ("2" in the figure) considers its eight neighbors at a distance of 2^{n-2}, and again selects the best match. After the final step, which uses eight neighbors at a distance of one pixel, the best match becomes the final motion vector.

The size of the search window for a given number of steps n is $W = 2^n - 1$, or

n can be determined for a desired window size by $n = \log_2(W+1)$ (assuming $W+1$ is a power of 2). The number of offsets C searched per block is equal to $8n+1$, or:

$$C = 8\log_2(W+1) + 1 \tag{2.13}$$

This requires much fewer operations than for the exhaustive search.

The logarithmic search method performs well if the block comparison metric increases monotonically around the optimal offset location [25]. If this is not the case, the logarithmic search can instead find a local minimum instead of the best match. For a large search range, the chances of finding a local minima are greater so the logarithmic search has a limit to its useful search window. Its performance can be worse than the exhaustive search due to this minima problem but the computational cost is significantly lower (e.g. 9 times lower for $W = 7$).

2.3.4 Hierarchical Search

Another method for reducing the number of search locations is the hierarchical search [25, 27]. An additional advantage of this method is that it increases the noise immunity of the motion estimation process, although it requires more storage for subsampled frames.

The algorithm uses the original and several subsampled resolutions of the current and reference frames, as shown in Figure 2.5. The image levels are numbered from 0 for the original (highest) resolution image, to the lowest level $N-1$, each level being a quarter of the size of the lower level. Motion estimation begins at level $N-1$ by dividing the current frame into $L \times L$ blocks and performing an exhaustive search. The next lower level $N-2$ is then divided into $L \times L$ blocks. Each block uses the estimate from the corresponding block in the previous level as a starting

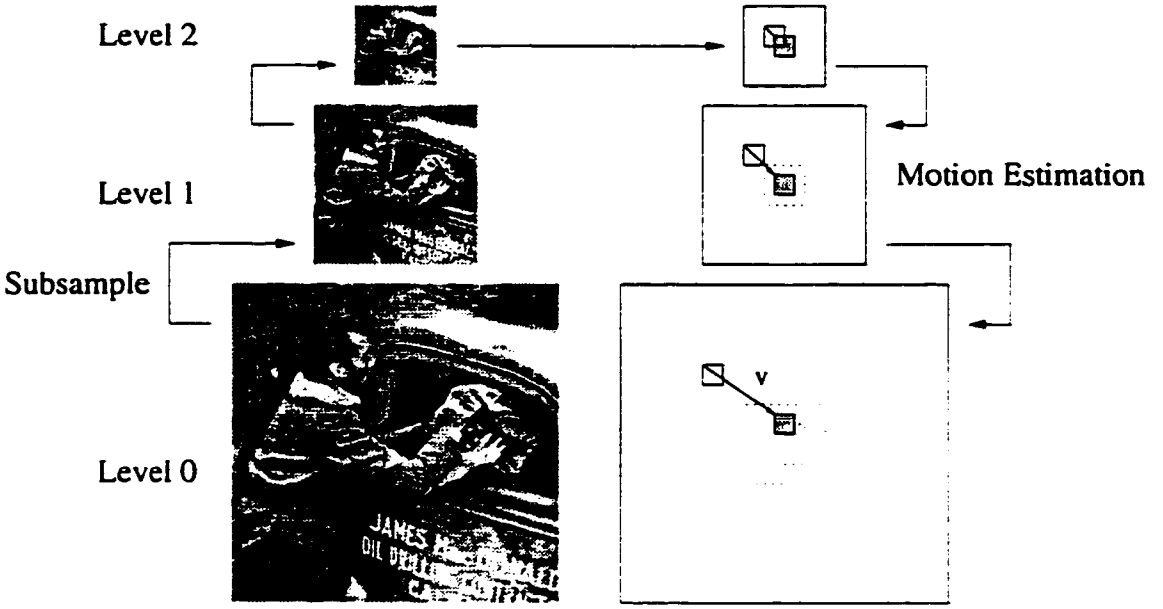


Figure 2.5: Hierarchical block search method.

point (zero offset) for its search. The best match estimated at level 0 is chosen as the best vector for that block. In this way, all $L \times L$ blocks in the full-resolution current frame are assigned a motion vector.

A search window w at level n scales to $2^n w$ at level 0 (full resolution). Because of this, very small search windows can be used at each level to search a large effective area. The search window at full resolution is $W = \sum_{i=0}^{N-1} 2^i w = w(2^N - 1)$, where w is the search window used at each level. The required number of block comparisons equals the comparisons at full resolution and all other resolutions. Assuming an exhaustive search within the window w at each level, $(2w + 1)^2$ comparisons are needed per block per level. The number of blocks decreases by a factor of 4 at each level, so the effective number of search locations C per full-resolution $L \times L$ block is:

$$C = \sum_{i=0}^{N-1} \frac{1}{4^i}(2w + 1)^2$$

$$= \frac{4}{3}(2w+1)^2\left(1-\frac{1}{4^N}\right) \tag{2.14}$$

Instead of an exhaustive search within the small window w at each level, methods similar to the full-resolution search techniques can be used to reduce the number of locations searched at each level. One such technique, the adaptive block matching search [13], selects five offsets around the start location, then searches at most three locations around the best match, as shown in Figure 2.6. A total

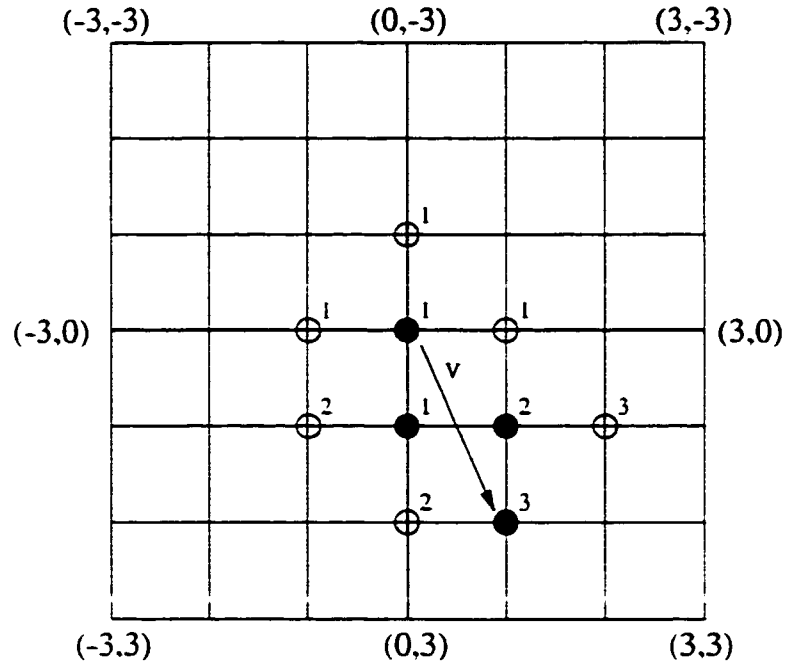


Figure 2.6: The adaptive block matching search method. Best matches are shown as dark circles.

of w steps can cover a window of size w. The locations searched per block per level is $5+3(w-1)$ in the worse case: the total number of search locations C per full-resolution block using the reduced search is:

$$C = \frac{4}{3}(5+3(w-1))\left(1-\frac{1}{4^N}\right) \tag{2.15}$$

The hierarchical search algorithm suffers from the same local minima problems as the logarithmic approach, particularly when using the adaptive block match-

ing search at each level. It also has the possibility of estimating incorrect offsets involving small objects which are eliminated by the subsampling process. Defects, however, are also reduced in size at lower resolutions so the algorithm is robust to noise. This method has the lowest computational cost of the algorithms reviewed.

2.4 Image Scaling

The method to create multiple frame resolutions, such as those used by the hierarchical motion estimation algorithm, deserves special mention. Each level is one-quarter the size of the previous level (width and height are half the previous level) and is created by filtering a higher resolution image, then decimating by two in the horizontal and vertical directions. The method for filtering the image should be a lowpass operation which removes all or most of the energy above one-quarter the sampling rate. This is necessary so that the decimation operation will not cause aliasing in the lower-resolution image.

A simple filtering method is four-pixel averaging [28], which is a two-dimensional FIR filter expressed as the difference equation:

$$y(i,j) = \frac{1}{4}(x(i,j) + x(i-1,j) + x(i,j-1) + x(i-1,j-1)) \qquad (2.16)$$

This method is very efficient and requires only 4 operations per low-resolution pixel. However, it passes a significant amount of frequencies above one-quarter the sampling rate, as shown by its frequency response in Figure 2.7. These frequencies will be aliased when the image is scaled to a lower resolution.

A Gaussian lowpass filter has been proposed as a better prescaling filter [3]. The filter kernel $g(i,j)$ is given as:

$$g(i,j) = w(i,j)\frac{1}{A}\exp^{-\left(\frac{r^2}{2\sigma^2}\right)}$$

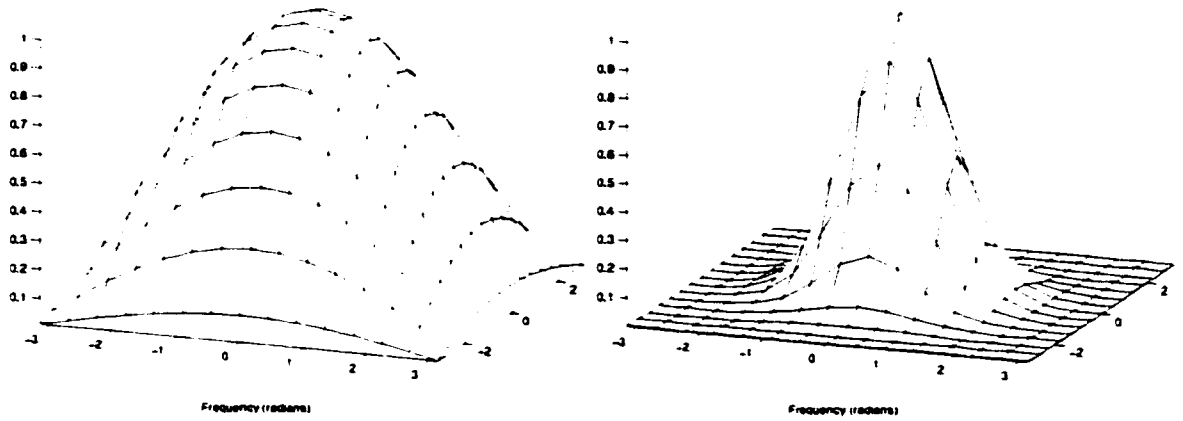


Figure 2.7: Frequency response of the averaging (left) and Gaussian (right) scaling filters.

$$
\begin{aligned}
r &= \sqrt{i^2 + j^2} \\
w(i,j) &= \begin{cases} 1 & \text{if } i^2 + j^2 \leq R^2 \\ 0 & \text{otherwise} \end{cases} \\
A &= \sum_{|i| \leq R} \sum_{|j| \leq R} w(i,j) \exp^{-\left(\frac{r^2}{2\sigma^2}\right)}
\end{aligned}
\tag{2.17}
$$

The size of the filter determines the radius R; for a 9×9 filter, $R = 4$. The variance σ^2 controls the cutoff of the filter, and for the work in this thesis it is set to 2. As seen in Figure 2.7, the Gaussian filter eliminates most of the frequencies above one-quarter the sampling frequency, so less aliasing will occur during scaling. An added advantage of the Gaussian filter is its circular shape, so motion in all directions is favored equally [3].

While the Gaussian filter appears to offer better characteristics, it requires much more computation than the four-pixel averaging filter. For $R = 4$, 41 multiplications and 40 adds are required[5], compared with only four operations for the averaging filter.

[5] Assuming filter kernel elements with $w(i,j) = 0$ are not included in the calculation.

2.5 Acceptance Thresholds

With all motion estimation methods, it is sometimes possible to calculate a false vector due to the effects of noise or because of the assumptions of the motion model. Often this occurs because a large area is self-similar, and several offsets can provide a good match to the current block. This can also occur in the case of hierarchical motion estimation as similar regions are moved closer together by the scaling process, resulting in incorrect motion estimates that propagate to higher resolutions [27].

One solution which was proposed to solve this problem is the use of an acceptance threshold [20]. The error calculated by a block comparison metric at the best match location, $E(\vec{v}_{n-1})$, is compared with the error at zero offset, $E(0)$. If the ratio $\frac{E(\vec{v}_{n-1})}{E(0)}$ is less than some threshold t_a, then the estimate is accepted: otherwise, the motion vector is reset to zero. For hierarchical motion estimation, this step would be performed at each level. Preference is given to zero motion offsets, which reduces estimation errors at low resolutions.

Another method for reducing errors in the hierarchical motion estimation algorithm is to use only small search windows at each level so offsets are forced to lie near zero. Larger full-resolution windows can be searched by increasing the number of levels rather than the window size at each level. The problem with this approach is that errors which propagate past the lowest resolutions may not be recovered using small search windows at higher resolutions.

2.6 Summary

Motion estimation is a crucial step in motion picture restoration because it adjusts for the effects of motion between frames and aligns objects along the temporal

axis. Among the possible techniques for performing motion estimation, the most appropriate for a real-time restoration system is a block matching technique, because it is capable of measuring large motion displacements efficiently. This is only possible by using methods to reduce the number of locations; a number of possible algorithms were described in this chapter.

For the hierarchical search method, frames must be filtered and scaled to lower resolutions. Some possible filters were described and their frequency response was compared. While the Gaussian filter offers the best filter characteristics, an averaging filter requires much less computation. Acceptance thresholds, which can increase the accuracy of the hierarchical search method, were also described.

Chapter 3

Detection of Film Defects

One method of concealing damage on a film frame would be to apply a suitable filtering operation (lowpass is one possibility) to all pixels in the frame. This would certainly decrease the effects of dirt and scratches, but it would also blur or distort undamaged portions of the frame. For this reason, it is desirable to first detect which pixels are likely to be damaged, and then restore only those areas containing damaged pixels.

While it would be possible to use a two-dimensional processing technique on each frame to find damage locations [28], this does not make use of the statistical properties of film damage. Scratch or dirt locations are unlikely to occur in the same location in consecutive frames. For this reason, it is more efficient to use information from surrounding frames to detect and conceal damage [15]. Two-dimensional processing would be accurate only if the algorithm could distinguish between a blotch location and, for example, a baseball. Because of the wide variety of damage types and objects, this is unlikely.

The "standard" method for film damage detection [3] uses the motion compensated frames $\hat{I}_{n-1}$ (previous) and $\hat{I}_{n+1}$ (next), as well as the current frame I_n

to detect defect locations within the current frame. An assumption is made that a dirt or scratch location within the current frame will not be damaged in the surrounding frames. As long as the motion estimation is accurate, damage locations in the current frame can be detected by comparisons with surrounding pixels in the motion compensated previous and next frames. The results can be represented as a bitmap, or "detection mask", which flags those locations in the current frame that are likely damaged.

A detection algorithm requires some threshold which will classify pixels as either damaged or undamaged. This setting is between two extremes: if it is set too low, almost all pixels in the current frame will be identified as damaged. If it is set too high, few or no pixels will be identified as defects, regardless of the actual amount of damage in the frame. In determining the threshold, a tradeoff must therefore be made between the allowed number of false detections and the number of missed detections.

This chapter contains a discussion of several types of detection algorithms and their computational cost. Spike detection methods, which use heuristics to determine damage locations, are discussed first. Next, a detector based on rank ordered differences is described, followed by a morphological detector. More complex detectors based on Markov random fields and autoregressive models are briefly discussed, but their high computational requirements make them unsuitable for real-time restoration. Finally, possible techniques for vertical line scratch detection are given, although the removal of line scratches is beyond the capabilities of the real-time implementation. The detection accuracy of these methods is evaluated on test sequences in Section 5.3.

3.1 Spike Detection Index (SDI)

The idea of a spike detection index (SDI) is based on the idea that a single pixel location (x, y) on several consecutive undamaged frames should have a relatively constant intensity, assuming the frames have been compensated for motion. A plot of intensity at that location as a function of time (or frame index) will be relatively flat. If a scratch or dirt spot appears in one frame, however, those affected locations will show a sharp "spike" in the intensity plot, as shown in Figure 3.1. The spike

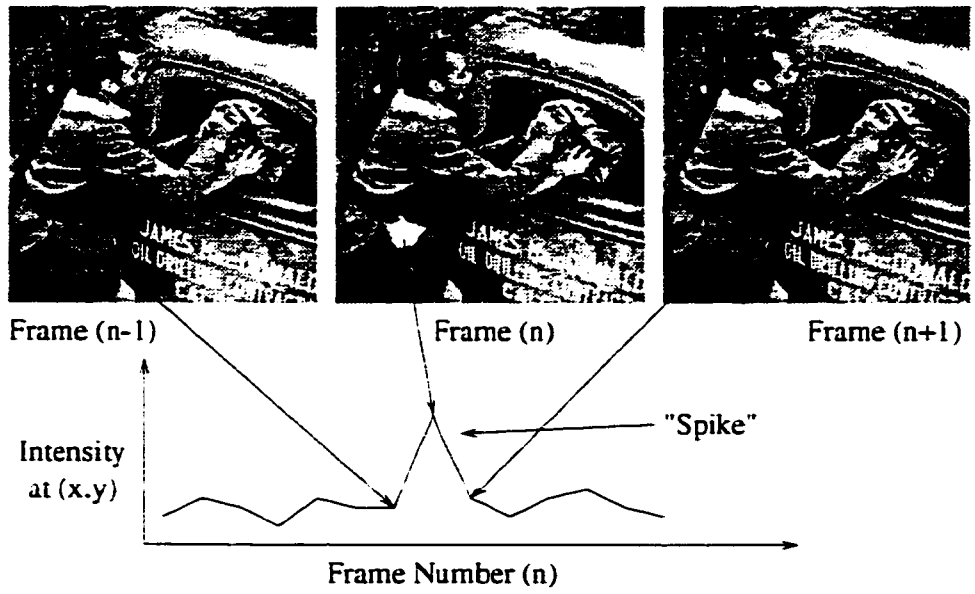


Figure 3.1: The spike detection technique for damage detection.

may be positive or negative, depending on whether the damage intensity is light (scratch) or dark (dirt).

The first to use spike detection for defect detection was Storey [11], although be did not use the name SDI. His detector did not make use of motion compensation because of the large amount of computation required. Among other heuristics for determining defect locations was the following formula, which uses the name SDIp

proposed in [3]:

$$\begin{aligned} e_b &= I_n(\vec{r}) - \hat{I}_{n-1}(\vec{r}), \qquad e_f = I_n(\vec{r}) - \hat{I}_{n+1}(\vec{r}) \\ D_{SDIp}(\vec{r}) &= \begin{cases} 1 & \text{if } (|e_b| > e_t) \text{ and } (|e_f| > e_t) \text{ and } (\text{sign}(e_b) = \text{sign}(e_f)) \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.1}$$

The equation has been extended to use motion compensation by the use of the frames $\hat{I}_{n-1}$ and $\hat{I}_{n+1}$. In its original form, I_{n-1} and I_{n+1} were used instead, and the detector performance suffered when applied to scenes with significant motion.

Another detector to use the spike detection index was presented in [29], which used the following formula:

$$\begin{aligned} e_1 &= |I_n(\vec{r}) - \hat{I}_{n+1}(\vec{r})|, \qquad e_2 = |I_n(\vec{r}) - \hat{I}_{n-1}(\vec{r})| \\ SDI(\vec{r}) &= \begin{cases} 1 - \left|\frac{e_1 - e_2}{e_1 + e_2}\right| & \text{if } (e_1 > t_1) \text{ or } (e_2 > t_1) \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.2}$$

The resulting $SDI(\vec{r})$ is a value between 0 and 1. A detection threshold t_s can be applied so that a pixel is flagged as a defect ($D_{SDI}(\vec{r}) = 1$) if $SDI(\vec{r}) > t_s$, and undamaged otherwise ($D_{SDI}(\vec{r}) = 0$). The other threshold t_1 is selected so as to avoid problems when e_1 and e_2 are close to zero.

The SDI performs well in situations where motion estimates are accurate, but it is very sensitive to situations where objects are covered or uncovered due to motion. To make the detector more robust to incorrect motion vectors, the modified spike detection index (SDIa) was proposed [20]:

$$\begin{aligned} e_b &= |I_n(\vec{r}) - \hat{I}_{n-1}(\vec{r})|, \qquad e_f = |I_n(\vec{r}) - \hat{I}_{n+1}(\vec{r})| \\ D_{SDIa} &= \begin{cases} 1 & \text{if } (e_b > e_t) \text{ and } (e_f > e_t) \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{3.3}$$

This detector is basically identical to the SDIp detector in Equation 3.1 without the constraint that the frame differences be the same sign.

It would be expected that both the forward difference e_f and the backward difference e_b would have the same sign for a genuine defect, assuming compensation for motion. In practice, however, this is not always the case. If a large defect in the current frame covers most of the motion block, the vector estimated by block matching will be essentially random because the best match for the *defect* will be found, not the obscured image data. The estimates in the forward and backward directions will not necessarily be the same. Thus, when the spike detection operation is performed, there may be large differences between the blotch pixels and the previous or next frame data, but these will not necessarily have the same sign. The SDIp detector, therefore, will miss these cases but the SDIa will detect some of them. In other situations with motion estimation errors, the SDIp may be more accurate. Comparisons are shown in Section 5.3.

The computational costs of these detectors are compared assuming any simple arithmetic or logical operator, comparison, or absolute value each counts as a single operation. The cost of accessing pixel values (i.e. reading the value from the frame buffer memory) is ignored. For the SDIp detector, the frame difference and absolute value must be calculated in both directions (4 operations), and the comparisons and logical operators require an additional 5 operations for a total of 9 operations per pixel in the worst case, where every conditional needs to be evaluated. The SDI requires 13 operations per pixel in the worst case, where e_1 or e_2 are greater than t_1 and each pixel is a defect. SDIa requires only 7 operations per pixel.

In spite of their simplicity, the heuristic spike detection index methods perform very well, are fairly robust to noise, and require very few operations to perform [20]. Their accuracy is limited by their purely temporal nature: using more pixels in the previous and next frames can improve performance, as seen in the next

section.

3.2 Rank-Ordered Difference (ROD)

The rank-ordered difference (ROD) detector attempts to improve on the performance of the spike detection methods by using more pixels in the previous and next frames in order to better characterize and identify defect locations. As described in [30], ROD uses three pixels in each of the previous and next frames, the locations of which are shown in Figure 3.2. The pixel values are represented by an

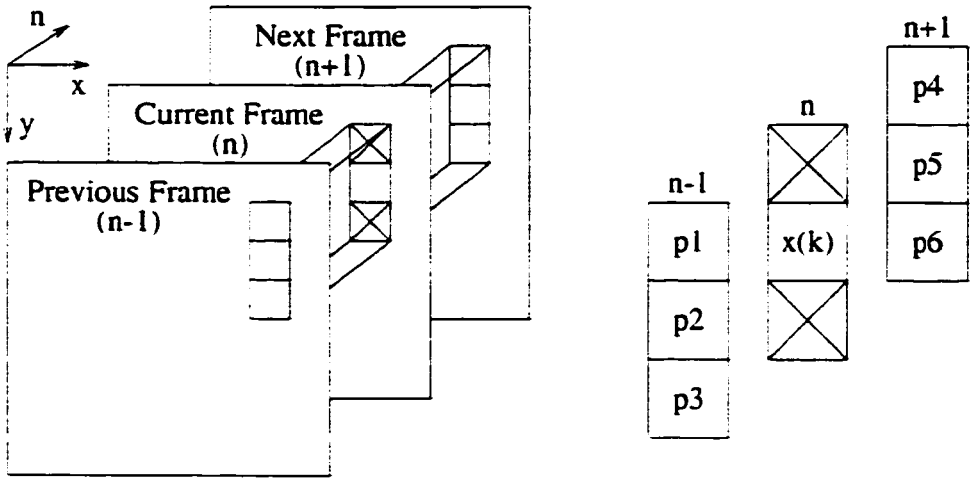


Figure 3.2: Pixel support for the rank ordered difference (ROD) detector.

array $p_i(\vec{r}), i = 1..6$ as follows, where $\vec{r}_{y-1}$ and $\vec{r}_{y+1}$ are shorthand for the vectors $(r_x, r_y - 1)$ and $(r_x, r_y + 1)$:

$$p_i(\vec{r}) = \left[\hat{I}_{n-1}(\vec{r}_{y-1}), \hat{I}_{n-1}(\vec{r}), \hat{I}_{n-1}(\vec{r}_{y+1}), \hat{I}_{n+1}(\vec{r}_{y-1}), \hat{I}_{n+1}(\vec{r}), \hat{I}_{n+1}(\vec{r}_{y+1})\right] \tag{3.4}$$

The values of $p_i(\vec{r})$ are sorted to form the vector $q_i(\vec{r})$, and the mean is calculated as

$$m(\vec{r}) = \frac{q_3(\vec{r}) + q_4(\vec{r})}{2} \tag{3.5}$$

Finally, the rank-ordered differences $d_i(\vec{r}), i = 1..3$ are calculated as:

$$d_i(\vec{r}) = \begin{cases} q_i(\vec{r}) - I_n(\vec{r}) & \text{if } I_n(\vec{r}) \leq m(\vec{r}) \\ I_n(\vec{r}) - q_{7-i}(\vec{r}) & \text{if } I_n(\vec{r}) > m(\vec{r}) \end{cases} \tag{3.6}$$

Three thresholds T_1, T_2, and T_3 are used, with $T_1 \leq T_2 \leq T_3$. If $d_i(\vec{r}) > T_i$ for at least one T_i, the pixel is marked as a defect. The main threshold is T_1, which controls the compromise between false and missed defects. The parameters T_2 and T_3 have less impact on the detection performance.

In calculating the number of operations to perform the ROD algorithm, the creation of $q_i(\vec{r})$ is assumed to use a simple sorting algorithm such as selection sort [31], which requires C_2^n operations. In this case, with six pixels, $C_2^n = 15$ operations. Calculating $m(\vec{r})$ requires two operations, another four operations to determine $d_i(\vec{r})$ (one comparison, three subtractions), and five more operations for threshold comparisons for a total of 26 operations per pixel in the worse case.

The ROD detector can offer better performance than spike detection due to its use of more support pixels [3, 30]. It requires more operations to compute, but it is still very efficient.

3.3 Morphological Detectors

A detector is presented in [13] which uses morphological operators within the current frame to identify regions which match the characteristics of film damage—small regions of relatively constant intensity, close to black or white, with sharp edges. The standard morphological operators, extended to greyscale images, are defined as:

$$\text{Dilation: } I_n(\vec{r}) \oplus B = I_n(\vec{r} + \vec{p}) : \max_{\vec{p} \in B}(I_n(\vec{r} + \vec{p}) - B(\vec{p})) \tag{3.7}$$

$$\text{Erosion:}\quad I_n(\vec{r}) \ominus B = I_n(\vec{r}+\vec{p}) : \min_{\vec{p}\in B}(I_n(\vec{r}+\vec{p}) + B(\vec{p})) \tag{3.8}$$

where B is a structuring element defined below. Morphological processing is done using only the current frame I_n. Dilation followed by erosion ("closing") effectively eliminates small dark regions, which are local minima. Therefore, the difference between the closed image and the original should have peaks at locations which are covered by dirt. In a similar way, erosion followed by dilation ("opening") eliminates bright regions.

The structuring element B must be chosen to accurately detect defect locations while not being fooled by ambiguous regions. The authors in [13] decided on a combination of two structuring elements: B_0, which was a 5×5 matrix of zeros, and B_n, also 5×5, which had the following form:

$$B_n = \begin{bmatrix} 2n & 2n & 2n & 2n & 2n \\ 2n & n & n & n & 2n \\ 2n & n & 0 & n & 2n \\ 2n & n & n & n & 2n \\ 2n & 2n & 2n & 2n & 2n \end{bmatrix}$$

where n was selected to match image and damage characteristics. The combination detects both small and larger defects. The detection of dirt and scratches is therefore performed by the formulas:

$$P_b(\vec{r}) = ((((I_n(\vec{r}) \oplus B_0) \oplus B_n) \ominus B_n) \ominus B_0) - I_n(\vec{r}) \tag{3.9}$$

$$P_w(\vec{r}) = I_n(\vec{r}) - ((((I_n(\vec{r}) \ominus B_0) \ominus B_n) \oplus B_n) \oplus B_0) \tag{3.10}$$

$$D_m(\vec{r}) = \begin{cases} 1 & \text{if } P_b(\vec{r}) > t_m \text{ or } P_w(\vec{r}) > t_m \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

P_b detects dark regions, and P_w detects light regions. If either exceeds the threshold t_m, the location is marked as a defect.

As a final improvement to accuracy, the morphological detector is combined with an inter-frame detector which behaves identically to SDIa. If a location is marked as a defect by both the intra-frame algorithm and the SDIa detector, it is accepted as a damage location.

The structuring elements B_0 and B_n are regular and symmetric, and this fact can be used to reduce the number of computations required to perform erosion or dilation across an image [13]. When the structuring element is applied to a new pixel, the maximum/minimum calculations only have to be applied to the newly covered pixels. Using this optimization, computation of an erosion or dilation by B_0 is reduced to 4 operations per pixel, and 21 operations per pixel for B_n, according to [13]. The calculation of P_b or P_w therefore takes a total of 51 operations. Calculating D_m requires 3 operations, and SDIa requires an additional 7 operations, for a total of 112 operations per pixel. This is significantly higher than both the spike detection methods and the ROD detector and beyond the abilities of the real-time restoration system, although the simplicity of its operations makes this method attractive for a custom hardware implementation.

3.4 Markov Random Fields (MRF)

The principle behind using Markov random fields for defect detection is to model the dirt and scratches as a separate frame, which lies between the previous (or next) and the current frame. By defining an appropriate "neighborhood", the algorithm can be made to favor connected defect regions. A complete discussion is given in [9]; the following description is only a brief overview, because the high computational cost makes it unsuitable for a real-time implementation.

Let the optimal defect detection frame be called D, where $D(\vec{r}) = 1$ for a

defect and 0 otherwise. The probability that a certain defect frame d is optimal can be expressed as the following *a posteriori* distribution [20]:

$$P(D=d) = \frac{1}{Z}\exp\left(\frac{-1}{T}\sum_{\vec{r}\in S}\alpha(1-d(\vec{r}))(I_n(\vec{r}) - \hat{I}_{n-1}(\vec{r}))^2 - \beta_1 f(d(\vec{r})) + \beta_2\delta(1-d(\vec{r}))\right) \quad (3.12)$$

The parameters α, β_1, and β_2 control the clustering of defect regions, and Z normalizes the distribution. The parameter T is used as a parameter for an annealing process. $f(d(\vec{r}))$ is equal to the number of neighbors surrounding location $\vec{r}$ with the same value as $d(\vec{r})$; valid neighbors are defined for some small square or circular region [20].

A simulated annealing process is applied to Equation 3.12 until it converges to a detection frame D. This technique is applied to the previous and current frame, then to the current and next frame. Locations which are marked as defects in both cases are accepted as being damaged.

The computational cost of the MRF detector is reported as 250 operations per pixel [20], assuming five iterations of the annealing process. Despite this large processing requirement, the performance of the MRF detector is almost identical to SDIa. The only case where MRF offers a significant improvement is in the detection of damage locations with poor contrast with the surrounding area, because of its connectivity characteristics [9]. In practice, however, these types of defects are uncommon and not very noticeable. The MRF method, in general, has no advantages over the spike detection algorithms despite its high computational cost, and it is therefore not considered further.

3.5 Autoregressive Models (AR)

A three-dimensional autoregressive model has been successfully applied to the blotch detection problem. Its high computational requirements and poor performance make it unsuitable for a real-time restoration system so only a brief overview is given here; a full discussion is given in [7].

The AR method creates a predicted image $P_n(\vec{r})$ using a weighted sum of pixels from a support region in the previous, current, and next frames [20]:

$$P_n(\vec{r}) = \sum_{k=1}^{N} a_k \hat{I}_{n+q_n(k)}(r_x + q_x(k), r_y + q_y(k)) \tag{3.13}$$

The support region, of size N, is defined by a vector $\vec{q}(k) = (q_x(k), q_y(k), q_n(k), k = 1...N)$ which represents the locations of support pixels in the current and surrounding frames. A least-squared estimation technique [32] is used to minimize the error $\epsilon_n = P_n - I_n$ by selecting the appropriate values a_k for the filter coefficients. An alternate technique is to create a forward prediction image P_{n+1} and a backward prediction image P_{n-1} using a separate adaptation process for each [20].

The predicted image P_n is compared with the current image, and any differences larger than a threshold are assumed to be defects. When two prediction images (P_{n+1} and P_{n-1}) are used, an error is detected when the threshold is exceeded for both. The number of model coefficients depends on how much support is desired in the current and reference frame: typically, using support only in the reference frame gives superior performance, because damaged pixel locations will usually be surrounded by other damage in the same frame [7].

The autoregressive model coefficients a_k must be estimated from the image sequence. In damaged sequences, the effects of dirt and scratches can bias this estimation process significantly, and therefore the detection accuracy decreases. As

shown in [20], the AR detector provides excellent results when the coefficients can be estimated from an undamaged version of the sequence to be restored (obviously impossible in a practical situation). When estimating coefficients from a damaged sequence, the detector performs badly—it essentially learns to reconstruct damage instead of the original image.

As reported in [20], the computational cost for the AR detector is 140 operations per pixel, assuming 8×8 pixel blocks and a support region of nine pixels in each reference frame. The performance of the algorithm is poor [20], especially when considering its large computational requirements. It is therefore unsuitable for a real-time restoration system.

3.6 Line Scratch Detection

Line scratches occur in frames when the film is abraded by particles of dust in the projector or telecine. They appear as (almost) vertical lines in the same location over several frames. As such, the algorithms used to detect random dirt and scratches will fail to detect most line scratches because they are not temporally local to a single frame. Various algorithms have been developed to deal specifically with line scratch detection [3, 9].

Since line scratches occur in almost the same location in adjacent frames, temporal information is unreliable for detection. Instead, two-dimensional algorithms are used which attempt to find locations resembling a line scratch. In the horizontal direction, the intensity across a scratch appears as an impulse surrounded by sidelobes of opposite intensity, similar to a damped sinusoid [3]. Detection methods therefore attempt to find locations which match this profile. Subsampling the image vertically improves detection by isolating only those vertical lines which cover the

entire height of the frame.

A detailed description of the methods employed to detect and remove line scratches is beyond the scope of this thesis, so only a basic overview is given here. A reversible-jump Markov chain statistical approach provides good results [9], as does a method based on the Hough transform [28, 3]. Removing the scratches can be done with a spatial filtering operation [3].

3.7 Summary

In this chapter, several methods for detecting random damage within motion picture films were discussed. The spike detection methods are computationally efficient and reasonably accurate, and the rank ordered difference detector improves detection accuracy at a slightly higher computational cost. A morphological detector combined with a spike detector has the potential to improve detection accuracy by using more spatial information, but its computational requirements are too high for real-time restoration on the current system. Detectors based on Markov random fields and autoregressive models are also too complex for real-time restoration. A brief discussion of possible methods for removing vertical line scratches was also given. Line scratches are not classified as random damage and removing them is beyond the scope of this work.

Chapter 4

Concealment of Film Damage

The final step in the motion picture restoration process is to conceal the damaged film locations (as marked by the detector). Damaged pixels must be reconstructed by using information from surrounding pixels, possibly within the current frame and/or in the previous and next motion compensated frames.

Some attempts have been made at global filtering operations, which are applied to the entire image frame without first detecting damage locations [33, 28, 7, 17]. While these methods can provide good results, they depend heavily on the quality of the motion estimation. When motion estimates are incorrect, unrelated image pixels (from surrounding frames) can be used in the filtering operation, resulting in incorrect restoration and visible artifacts. If the damage detection phase can be done quickly and efficiently, it is worthwhile to perform.

In the following sections, several methods for reconstructing damaged image pixels are described. All are intended to be applied only at those locations marked as damaged by the detector, and therefore undamaged regions are unaffected. The previous and next frames are assumed to have been compensated for motion. Simple averaging and median filters are described first, followed by more complex methods

based on Markov random fields and autoregressive models. The more complex techniques require far too much computation to be performed in real-time and are therefore unsuitable for the current restoration system.

4.1 Averaging Filters

A simple averaging approach can be used to reconstruct damage locations [28]. The surrounding pixels in the current and/or surrounding frames are averaged to provide an estimate of the original pixel intensity at the defect location. This assumes that motion compensated frames have high temporal correlation, and that intensity varies gradually within the support region in a single frame. Within these constraints, the average is a good approximation of the original pixel value.

The use of a support region for the averaging operation uses the notation *a:b:c*, which indicates *a* support pixels in the previous frame, *b* in the current, and *c* in the next frame [15]. Several possible support regions are shown in Figure 4.1. The

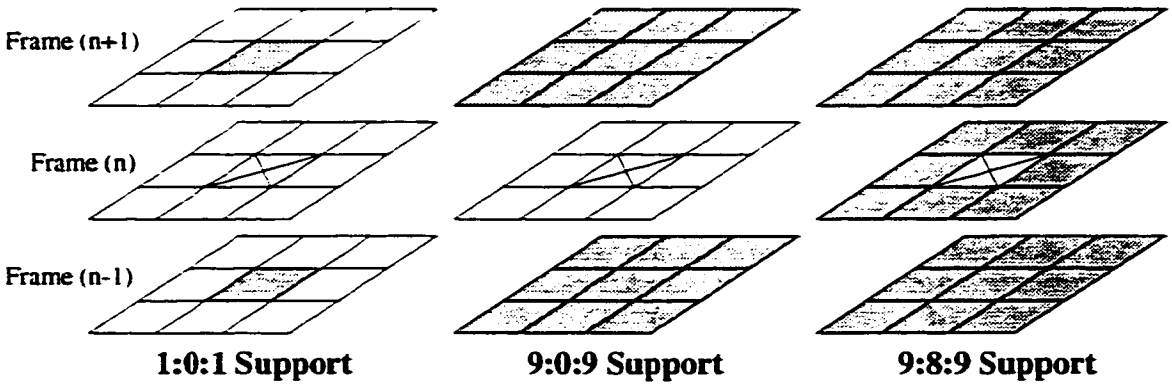


Figure 4.1: Example support regions for the averaging filter.

pixel to be restored is indicated by an "X", and all shaded pixels are averaged to provide the reconstructed intensity value. For instance, the 1:0:1 averaging operation

calculates the interpolated pixel value $\tilde{I}_n(\vec{r})$ as:

$$\tilde{I}_n(\vec{r}) = \frac{1}{2}\left(\hat{I}_{n-1}(\vec{r}) + \hat{I}_{n+1}(\vec{r})\right) \tag{4.1}$$

Larger support regions include more pixels from the surrounding area (spatially and temporally) with the intention that more pixels will provide a better approximation for the damage location. Note that Equation 4.1 uses only the unrestored frame data—essentially a three-dimensional FIR filter.

In the simplest case, all pixels from the support region are used when calculating the average. However, when frames are processed in sequence the damage locations for the previous and current frames have already been identified. These defects will introduce errors in the averaging operation. A more sophisticated approach is to only use undamaged pixels within the support region when calculating the average.

The averaging operation is very efficient to calculate, requiring $a + b + c$ calculations for an *a:b:c* support region. This calculation gives 2 operations per damaged pixel for 1:0:1 support, 18 for 9:0:9, and 26 for 9:8:9. The computation increases by a factor of two if only undamaged pixels are used, since the detection mask has to be checked for each pixel in the support region.

Averaging performs well in areas of constant or slowly varying intensity, but larger support regions can blur the reconstructed region in highly textured areas. Using an averaging operation with less support, such as 1:0:1, eliminates this problem but is more susceptible to motion estimation errors. Support regions larger than 9×9 pixels generally lead to worse results for more computation, since there is a greater chance that the image is not smooth within that region.

The averaging operation, as viewed as a three-dimensional FIR filter, assigns equal weights to all pixels in the support region. If only undamaged pixels are used,

weights of zero are assigned to damaged pixel locations. Interpolation quality could possibly improve by assigning varying weights to support pixels. This process is used by the autoregressive concealment method described in Section 4.4, and so is not examined here.

4.2 Median Filters

When dealing with image sequences containing impulsive noise, a common practice is to use a median filtering operation [28]. Of all the pixels in the support region, the median is calculated and substituted for the damaged pixel location. Where the averaging filter will blur textured regions, the median filter can reconstruct some approximation of the underlying pattern due to its nonlinear nature [3].

The same notation as for the averaging filter is used for support regions—*a:b:c* indicating the number of pixels supported in the previous, current, and next frames. However, the damaged pixel is usually included in the median calculation, since if it is significantly different than the surrounding pixels it will not be chosen as the median, and therefore be ignored. In this way the median operation is more tolerant to false detections: a location marked as damaged may retain its original value if it is not extreme compared to other pixels in the support region.

Some possible configurations for support regions are shown in Figure 4.2, which are similar to those used in the averaging operation, except the center (damaged) location is included. With larger support regions, the median filter is better able to determine the "most common" pixel intensity. However, if by chance several pixels in the support region (current or surrounding frames) are also damaged, the output of the median filter may be inaccurate. The strategy mentioned in the previous section, where only undamaged pixels within the support region are used,

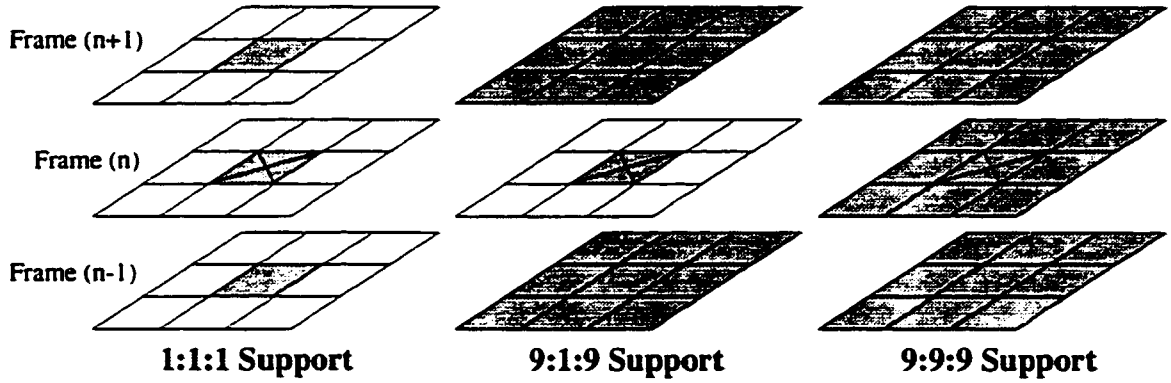


Figure 4.2: Example support regions for the median filter.

can also be applied to the median operation. It is less important, however, since damaged pixel values will likely be outliers compared with valid pixels, and will not be selected as the median.

An improved, multilevel median filter (ML3Dex), which is better able to reconstruct pixels when the surrounding area is damaged (all pixels in the support region are used, regardless of damage), is presented in [7]. It uses five different

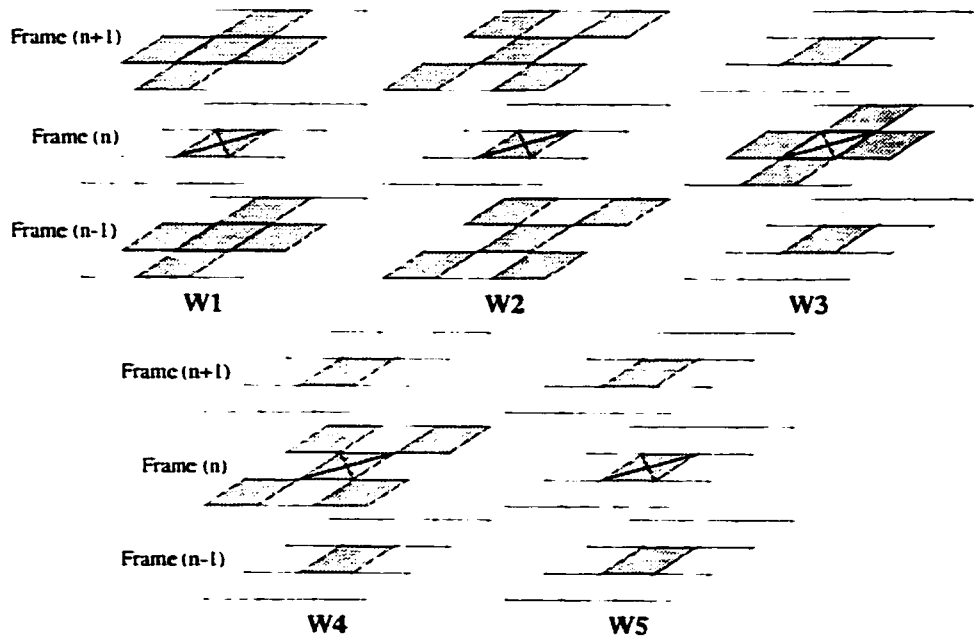


Figure 4.3: Masks for the ML3Dex multilevel median filter.

support regions (shown in Figure 4.3) per damage location, and calculates the final output as:

$$\begin{aligned} z_i &= \text{median}(W_i), i = 1..5 \\ \tilde{I}_n(\vec{r}) &= \text{median}(z_i), i = 1..5 \end{aligned} \tag{4.2}$$

That is, the median for each region is calculated, then the median of these is determined to be the reconstructed pixel value. By using several support regions in different configurations, the filter can adapt to situations in which several pixels in the current or surrounding frames are damaged. This is due to the fact that the regions place different weights (i.e. more or less pixels are considered) in the current and surrounding frames in each configuration.

The computational cost of the median operations can be calculated by assuming that the median of m values is calculated using a C_2^m operation sorting algorithm, such as selection sort [31]. For the single region median filters in Figure 4.2, $m = a+b+c$, which gives 3 operations per damaged pixel for 1:1:1 support, 171 for 9:1:9, and 351 for 9:9:9. The ML3Dex filter requires a total of 165 operations per damaged pixel to calculate the median of all masks and the final result.

Because of their nonlinear nature, median filters are able to reconstruct an approximation of textured regions without blurring. However, in some regions median filtering can cause artifacts due to the fact that it "rearranges" pixels from the surrounding area in reconstructing defect locations [15]. This is most evident in areas of fine detail, such as text. Using a median filter with purely temporal support, such as 1:1:1, can decrease these effects at the cost of being more susceptible to noise.

4.3 Markov Random Fields (MRF)

The Markov random field interpolator uses the same principles as the MRF detector described in Section 3.4, but in this case it attempts to reconstruct the original frame instead of a damage mask. It requires a large amount of computation [15] and therefore is unsuitable for real-time restoration. A brief overview is given here; complete details are in [9].

The MRF method determines the probability that a possible frame i_n matches the optimal reconstructed frame I_n and refines the estimate for i_n accordingly. An expression for this probability is [15]

$$p(I_n = i_n) = \frac{1}{Z_l} \exp\left(\frac{-1}{T} \sum_{\vec{r}:d(\vec{r})=1} \left[\sum_{\vec{s}\in\mathcal{N}} (i_n(\vec{r}) - i_n(\vec{r}+\vec{s}))^2 + \lambda \sum_{s_n\in\mathcal{T}} (i_n(\vec{r}) - i_{n+s_n}(\vec{r}))^2\right]\right) \tag{4.3}$$

The detected damage locations are represented by d, where $d(\vec{r})$ is equal to 1 if location $\vec{r}$ is damaged and 0 otherwise. $\mathcal{N}$ and $\mathcal{T}$ represent the spatial and temporal neighborhoods, respectively, for the MRF model. λ is the temporal weighting factor and Z_l normalizes the distribution. Convergence of i_n is reached by an annealing process using the parameter T.

The MRF interpolator effectively conceals damage and typically outperforms the averaging or median filtering methods [9], but its computational requirements are prohibitively high—about 22000 operations per damaged pixel [15]. This method was not therefore not considered for a real-time implementation.

4.4 Autoregressive Models (AR)

Like the autoregressive detector described in Section 3.5, the autoregressive interpolator adapts filter coefficients so as to predict the reconstructed image over a small area. The expression for predicting the reconstructed image is given as [15]:

$$I_n(\vec{r}) = \sum_{k=1}^{N} a_k \hat{I}_{n+q_n(k)}(r_x + q_x(k), r_y + q_y(k)) + \epsilon(\vec{r}) \tag{4.4}$$

As for the AR detector, the vector $\vec{q}(k)$ lies in the support region N in the current and surrounding frames. The error ϵ represents the difference between the predicted frame and the actual frame I_n.

In order to reconstruct the image data at damaged locations, the filter coefficients a_k must be predicted. A small area in the surrounding frames (presumably undamaged) is chosen to adapt the coefficients. They are then applied on the current frame to reconstruct damaged areas.

The AR interpolator gives excellent performance on damaged sequences [16, 7]. However, its computational cost is approximately 20000 operations per damaged pixel under typical conditions [15]. This is far too high to consider for real-time implementation.

4.5 The JOMBADI Algorithm

The JOMBADI (**JO**int Model **BA**sed **D**etection and **I**nterpolation) algorithm is a recent development in motion picture restoration [3]. It is not strictly an interpolator, because it combines damage detection and concealment in one step. It also implicitly refines motion estimates for optimal reconstruction.

Basically, JOMBADI requires an initial motion field and detection mask and these can be created by any simpler methods. It then randomly adjusts motion

vectors for some blocks and calculates a prediction for the reconstructed frame by creating a statistical model of the image data. This process continues until the results converge to some criterion (prediction error or maximum number of iterations). Full details of the method are given in [3].

Computational requirements for JOMBADI are much larger than for any of the other methods, even considering that it performs all restoration steps simultaneously. Because of its excellent restoration quality, it represents the best restoration algorithm currently available. Unfortunately, its high computational cost makes it unsuitable for real-time execution and therefore it cannot be used for this work.

It is interesting to note that while the JOMBADI algorithm detects damage much more accurately than other methods, the difference between their interpolated output is much less significant [3]. This suggests that an appropriate concealment method can compensate for detection errors, which is also supported by the results of Section 5.4.

4.6 Summary

In this chapter, various algorithms for concealment of film damage have been presented. The simplest are the averaging and median filters, which can be adjusted in terms of quality and computational cost by setting the size of their support region. More advanced methods based on Markov random fields and autoregressive models provide better restoration quality but at a much higher computational cost. Finally, the JOMBADI algorithm provides the best results at the highest complexity.

Chapter 5

Software Design of the Restoration System

In the preceding chapters, various algorithms for motion estimation, damage detection, and concealment have been described. In order to create a real-time, automatic restoration system, the algorithms that offer acceptable performance and computational efficiency must be chosen. This section describes the selection of the algorithms most appropriate for real-time implementation on the test architecture, a Precision MX TMS320C80 digital signal processing card.

All algorithms were first implemented in simulations which allowed easy selection of the many possible algorithms and parameters. Digitized test sequences WESTERN and MOBCAL (see Section 1.4) containing artificial damage were used as input to the restoration algorithms. The results were analyzed through a series of test sets which calculated accuracy of detection and other performance metrics using the restored output and the original undamaged sequences. New algorithms could be quickly implemented and evaluated for accuracy with this system, and the most promising were rewritten and optimized for the MX.

The discussion in this chapter starts with a description of the real-time implementation platform, the Precision MX, and a description of the TMS320C80 digital signal processor. Understanding the capabilities of this system is critical to selecting an algorithm which uses computing power and memory bandwidth efficiently for a real-time restoration implementation. The following sections contain descriptions of the comparisons performed between various algorithms for motion estimation, defect detection, and damage concealment. All algorithms were evaluated using simulations: the real-time implementation of the best algorithms is described in the next chapter.

5.1 Implementation Platform

The platform selected for implementation of the real-time restoration system was the Precision MX, a digital signal processing (DSP) card from Precision Digital Images Corporation [34]. It includes a 40 MHz Texas Instruments TMS320C80 digital signal processor [35], local on-card memory, and video capture and display hardware, as shown in Figure 5.1. The card fits in a personal computer (PC) PCI

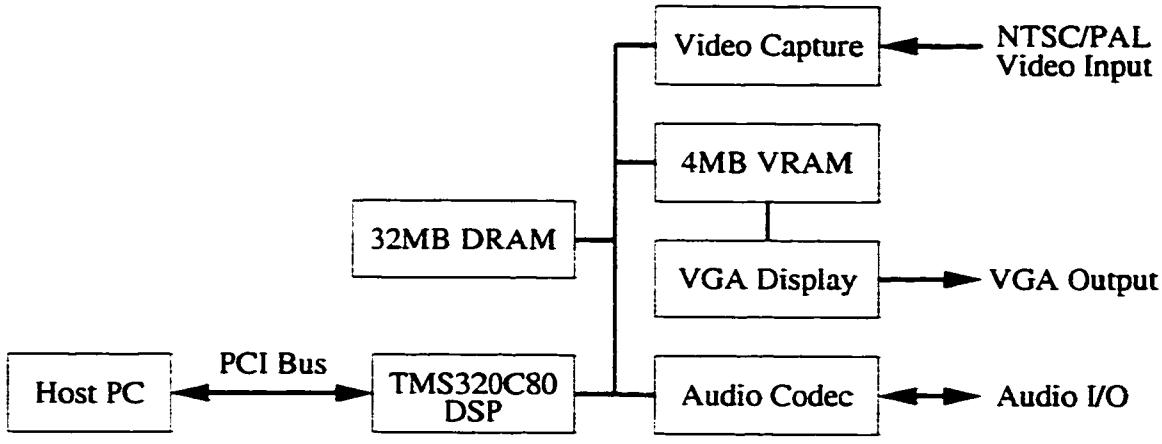


Figure 5.1: Components of the Precision MX DSP card.

bus slot and can send and receive data between on-card components and the host PC. It is capable of capturing video, processing the frame data, and displaying the processed frames in real-time. Video input is NTSC or PAL format video, and the output is a VGA signal which can be displayed on a computer monitor.

The MX was chosen for this application because it provides a flexible and powerful platform for a software-based restoration system. The TMS320C80 is optimized for video and graphics processing on small data units (8 or 16 bits). Creating custom hardware for this project would have been possible and likely would have superior performance, but would be difficult to modify to implement various algorithms. The Precision MX offers an excellent compromise between the speed of custom hardware, through the use of a video-optimized DSP, and the flexibility of a completely software implementation.

5.1.1 TMS320C80

The core of the MX card is the Texas Instruments TMS320C80 digital signal processor [35] (hereafter referred to as the C80). It is optimized specifically for the types of operations used in image and video processing. On a single chip (see Figure 5.2), the C80 includes five separate processors: a master processor (MP) and four parallel processors (PP), as well as internal memory, a transfer controller (TC) to move data to and from internal memory, and a video controller (VC) which generates timing for video capture and display. Components are connected with each other through an internal crossbar switch network.

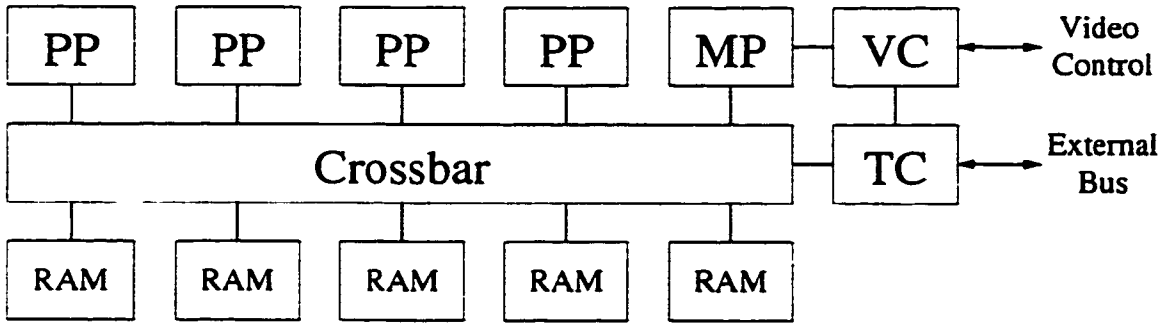


Figure 5.2: TMS320C80 internal layout.

5.1.1.1 Master Processor (MP)

The master processor (MP) is responsible for controlling all of the other parts of the C80 [36]. It is a fairly standard RISC processor, including floating-point arithmetic and bit-manipulation hardware. The MP uses both an instruction and data cache.

In a typical application, the MP would receive data (through the transfer controller) from the capture hardware or the PC and perform some preprocessing, then distribute the remainder of the work to the parallel processors. The PPs would signal when their work was complete, and the MP would recombine the data. For example, in a video decompression application, the MP would receive the compressed bitstream, decode the Huffman-encoded data, and pass the macroblocks to the PPs for processing.

The MP is intended mostly for high-level tasks, and not intensive calculation. The PP design is far better suited for the majority of video processing operations.

5.1.1.2 Parallel Processors (PP)

The parallel processors (PP) are responsible for the majority of computation in the C80 [37]. There are four independent PPs, each using its own instructions and data. Unlike the MP, they are optimized for fast integer operations, and do not include

floating-point hardware. They do, however, include facilities for zero-overhead loops and multiple instructions per cycle.

Within a single clock cycle, a PP is capable of concurrently executing a multiply, arithmetic/logical instruction, and two (on-chip) memory accesses. This is in specific special cases; depending on the registers or types of instructions used, fewer concurrent operations may be performed. Using concurrent execution, a PP can process data, store data from the previous operation, and load data for the next operation all in the same clock cycle.

The PP can also perform multiple arithmetic by splitting its 32-bit registers into two 16-bit words or four 8-bit bytes. When two registers are added (for example), each pair of words or bytes are added independently and set separate result flags (for overflow, zero, etc.). Multiple arithmetic can be performed for addition, subtraction, multiplication, boolean, and comparison operations. In this way, small data units (such as one-byte pixels) can be processed very quickly in parallel.

Hardware is included in the PP for zero-overhead loops, where the start and end addresses and the initial loop count are loaded into the loop hardware, and the correct number of loop iterations occur without the need for explicit loop instructions. Most operations can be made conditional depending on flag status, so conditional loads and ALU operations are possible without the need for if/then loop structures. Using loop hardware and conditional instructions eliminates the need for most branches.

5.1.1.3 Transfer Controller (TC)

The transfer controller (TC) is responsible for moving data between external and on-chip memory [38]. It is automatically initiated to perform cache loads for the

MP and PP, or transfer data for the video controller. While it can service single requests for external data, it is much more efficient to use "packet transfers" for this purpose. All memory transfers, internally and externally, are 64 bits in size.

A packet transfer can be initiated by the MP or PP to move memory between two locations (internal or external memory). The parameters of the transfer are stored on-chip in a "packet" structure, which contains the length and format of the source and destination, and other parameters relating to how the transfer is to take place. Three-dimensional data can be transferred using an "A" length, a "B" length and pitch, and a "C" length and pitch which may be different for source and destination: "A", "B", and "C" correspond to x, y, and z when dealing with 3D data. By making the source and destination lengths and pitches different, data can be rearranged as it is moved. Packet transfers have a priority selected when they are first issued, which determines the transfers that execute first.

5.1.1.4 Local Memory and Crossbar

The C80 contains 8 kilobytes of on-chip memory for each PP, divided into three independent 2 kilobyte banks and a 2 kilobyte instruction cache [35]. The MP uses 4 kilobytes each for instruction and data cache. PP instruction caches are divided into four 512 byte blocks, each of which hold 64 PP instructions. Each block holds contiguous data from a 512 byte aligned address. It is desirable to group PP code into 64 instruction units to utilize the cache most efficiently. When instructions or data are requested that do not lie in the on-chip cache, the PP or MP stalls until the TC can load the desired information from external memory, first writing back any changed cache data.

An internal 64-bit crossbar switching network routes data from any unit (PP,

MP, TC, etc.) to any memory bank. If two units wish to access the same memory bank in the same cycle, the lower priority unit must wait for the higher priority unit to finish. It is therefore important to divide on-chip memory among units to avoid contention. For example, the PP should process data in one memory bank while the TC fills another bank, otherwise the two units will compete for access and slow execution.

5.1.2 MX Support Hardware

The MX includes hardware to capture and display video data. Input is in the form of an NTSC or PAL video signal, at a maximum resolution of 640 × 480 pixels, 30 frames per second (for NTSC) [34]. Pixels are digitized as 8-bit greyscale, 16-bit YUV color, or 24-bit RGB color. Display is in the form of a VGA video signal, at a maximum resolution of 1600 × 1200 pixels at 60 frames per second. Display pixels can be in 8-bit greyscale or pseudo-color format, 16-bit color, or 24-bit RGB color.

The capture hardware is configured with the required resolution and color depth, and stores digitized lines of video into a FIFO buffer. At the end of each line, an interrupt signals the C80 that the buffer must be read into external memory.

Frames to be displayed must be stored in video ram (VRAM) so they can be transferred to the display hardware. The C80 video controller (VC) is programmed to provide the timing for the display and the video DAC [39].

The MX resides in a PCI slot in the host PC. Through the PCI bus, the PC is able to send code to the MX and control its execution by sending commands and data. The MX contains a PCI interface buffer which it uses to communicate with the host PC.

Audio capture and playback functions are supported by the MX to play and

record stereo 16-bit audio. A bidirectional FIFO provides the interface between the C80 and the audio codec chip. The restoration software does not make use of the audio capabilities of the MX.

5.2 Motion Estimation

As stated in Chapter 2, motion estimation is a critical factor in the performance of a restoration system. The estimation process should be accurate and robust to noise, since the input sequences contain significant damage in the form of dirt and scratches. Various methods for motion estimation were compared on the basis of their performance and suitability for real-time implementation on the C80.

The block matching method was chosen over the gradient or phase correlation methods because of its simplicity and large search area. The gradient method may require fewer operations, but the maximum estimation range is too small to be useful in video sequences. Phase correlation requires a large amount of computation. Another advantage for block matching is that it uses only integer operations, and it can efficiently take advantage of the C80 parallel processor multiple arithmetic instructions. Block matching was therefore chosen as the preferred motion estimation method without a quantitative comparison: the other methods are simply infeasible for real-time implementation so they were ignored.

There are many variations of the block matching estimation method, including ways for reducing the search space, comparison metrics, and block size [25]. The following sections compare several alternatives on the basis of performance and computational cost, by processing the WESTERN and MOBCAL sequences. These sequences are both artificially damaged, and contain various types of motion and frame information—WESTERN is perhaps more like a typical damaged film, while

MOBCAL contains complex objects and motion to test the robustness of the algorithms. Methods which performed well on both sequences were assumed to work well on most damaged films.

Most comparisons in the following sections show that the relative performance of several techniques were similar, but there was a larger difference in terms of computational cost. While other sequences may cause small relative differences in performance, the computational cost remains constant. The algorithms selected, therefore, were not necessarily those that offered the maximum performance, but those which had a high performance to computation ratio. Furthermore, when the algorithms were applied to actual degraded sequences (Section 6.3.1), the results were of the same scale as shown by the measurements in this section. The relative performance between the two test sequences was also different: in general, the algorithms performed better on WESTERN because it contains simpler objects and motion, but the relative performance of the algorithms on a single sequence was consistent.

Performance Metrics

There are several possible methods to compare the performance of motion estimation methods. Popular techniques are to use the entropy of the prediction errors [26] or the error between the original and predicted frames [21]. For applications in motion picture restoration, these techniques are inappropriate—the sequence used for estimation is damaged, so the optimal motion estimate will not necessarily correspond to the match which provides the lowest error.

Several metrics were tried for motion estimation comparisons, using the fact that the WESTERN and MOBCAL sequences contain synthetic damage and the

original undamaged sequences were available. One possibility was to perform motion estimation on the undamaged sequence and compare the difference between these vectors and those estimated from the damaged sequence. Another possibility was to perform motion estimation on the damaged sequence and use those estimates to determine error in the undamaged sequence. In practice, both of these methods showed little correlation between their relative performance and the performance of the actual restoration process (detection and concealment accuracy).

The reason why these metrics failed is they did take into account the fact that the restoration process actually uses a nonlinear combination of three frames. A combined error of the forward and backward estimation could be used to simulate this relationship, but this would still assume that the match with the lowest error is the best match. The restoration algorithms do not necessarily require this: they can perform well if the match contains a similar structure to the current block, within the detector thresholds.

From this analysis, the best motion estimation comparison metric was determined to be the performance of the defect detector. If the damage locations have been accurately identified, it is reasonable to assume that concealment will also be accurate. The SDIa detector was used for its simplicity and speed: the relative performance of the algorithms was the same regardless of the detector used. Performance was given as plots of percentage of correct detection locations (r_c) versus false detection locations (r_f) for a number of detector thresholds, as has been commonly used by other researchers [20, 30]. For an image of size $N \times N$, with a known number of damaged pixels N_d, the detector will identify a fraction C_d correctly and F_d incorrectly. The fractions r_c and r_f are calculated as [3]:

$$r_c = \frac{C_d}{N_d}$$

$$r_f = \frac{F_d}{N^2} \tag{5.1}$$

The correct detection locations are known by comparing the artificially damaged sequence with the original. High correct detection rates with low false detection are most desirable (upper left on the plots). For the plots, r_c and r_f were averaged over all frames in the sequence except for first and last, because these biased results due to the lack of previous or next frame information. A correct detection rate between 0.75 and 0.85 was generally a good tradeoff between correct and false detection.

5.2.1 Block Search Method

The most significant aspect to consider for motion estimation was the method for searching the motion window. Section 2.3 listed several algorithms for covering the search space which have greatly different computational requirements. Figures 5.3 and 5.4 show the results from applying these search methods to the WESTERN and MOBCAL sequences. The figure shows the percentage of correct detections versus false detections when using the SDIa detector, varying the threshold from 0 to 255 to create a curve for each motion estimation technique. Values with high correct detection and low false detection are most desirable.

Shown in Figures 5.3 and 5.4 are the results for the exhaustive, overlapped, logarithmic, and hierarchical search techniques with full-resolution search windows of $W = 7$ and $W = 15$. Detector performance with no motion estimation is also shown for comparison. The block size was chosen to provide the optimal performance with each method, which was $L = 16$ for the exhaustive and logarithmic methods, $L = 8$ for hierarchical, and overlapped search used a block size $L = 4$ with an overlap size $M = 16$. All methods used the MAD block comparison metric and had their parameters set to provide the same maximum vector offset at full resolution:

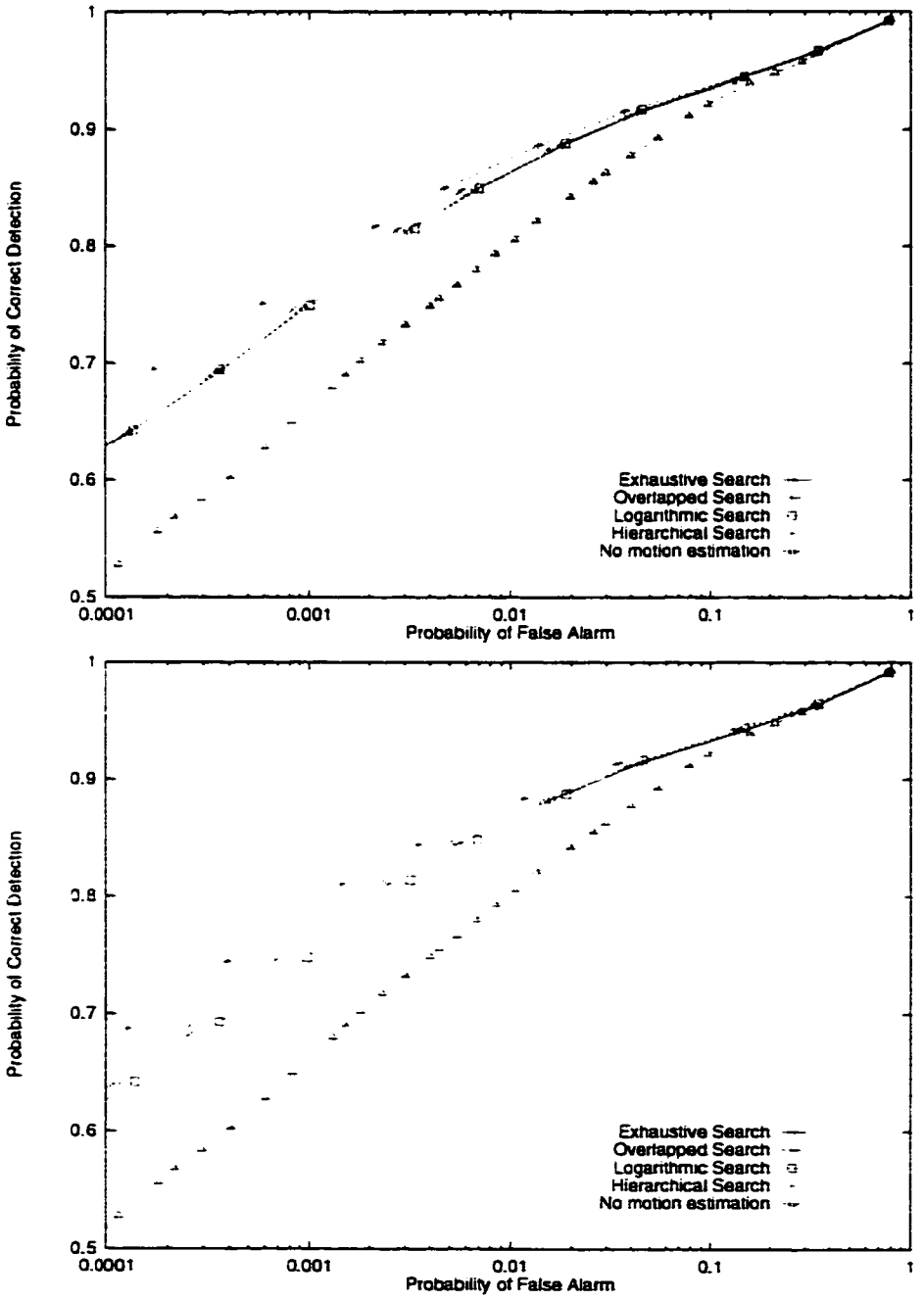


Figure 5.3: Comparison of block search techniques on the WESTERN sequence, with a full-resolution search extent of 7 pixels (top) and 15 pixels (bottom).

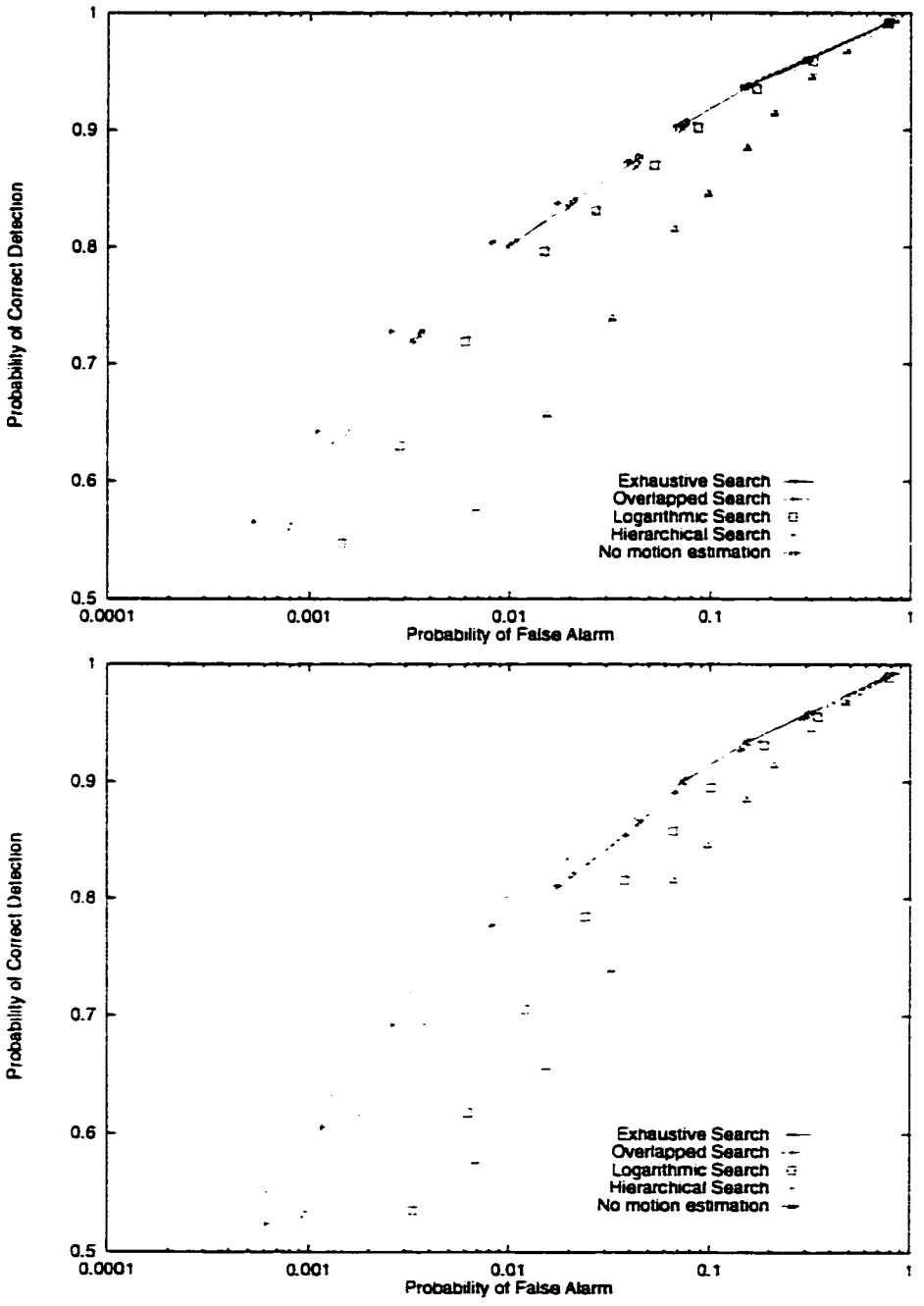


Figure 5.4: Comparison of block search techniques on the MOBCAL sequence, with a full-resolution search extent of 7 pixels (top) and 15 pixels (bottom).

for $W = 7$, logarithmic used 3 steps and hierarchical used 3 levels created by an averaging filter, with an exhaustive search at each level in a window of size $w = 1$; for $W = 15$, 4 steps/levels were used.

The curves of detector performance are indicative of the relative accuracy of the motion estimation methods. The overlapped method performed significantly better than the other methods in most cases, due to its ability to create a finer resolution vector field while maintaining noise immunity, although at a high computational cost. Methods which reduce computation, such as logarithmic and hierarchical search, had nearly the same performance as an exhaustive search on the WESTERN sequence, at much lower computation. In the MOBCAL sequence, the logarithmic approach performed poorly in comparison with exhaustive search, especially with a larger search window, due to its susceptibility to falling into local minima. Interestingly, the hierarchical search performed better than exhaustive search in the MOBCAL sequence, with the difference increasing with a larger search window. This was likely due to the natural noise filtering which occurs when the hierarchical frame resolutions were created: large defects at full resolution became small or were eliminated at lower resolutions. All motion estimation methods provided a significant improvement over no estimation.

The number of comparisons per block for all methods shown is given in Table 5.1, using the equations indicated and the parameters described earlier. Computation

Search Method	Equation	Comparisons per Block	
		($W = 7$)	($W = 15$)
Exhaustive	2.11	225	961
Overlapped	2.12	3600	15376
Logarithmic	2.13	25	33
Hierarchical	2.14	11.8	11.95

Table 5.1: Computation comparisons for block search methods.

for the exhaustive and overlapped search methods is far greater than for the other methods. It is fortunate that the hierarchical search, which requires the fewest operations, also provides excellent performance, second only to the overlapped search.

Selecting the best search method would be a choice between the overlapped search, which had the best performance but an enormous computational cost, and the hierarchical search, which had slightly worse performance but required much fewer operations. When designing a real-time restoration system, speed is critical and the overlapped method requires far too much computation for real-time execution, although it might be suitable for a hardware implementation. Therefore, the hierarchical search method was chosen as the best choice for restoration and the following sections compare other motion estimation parameters based on this search method. Several other researchers have also selected hierarchical search as the most appropriate for restoration applications [20, 4, 30].

5.2.2 Block Comparison Metric

The block search method specifies which locations are to be searched within the motion window, and blocks at these locations are compared using a block comparison metric. A good metric will compute a minimum when the motion vector is the best match. Several comparison metrics (described in Section 2.2) were used with the hierarchical detector on the WESTERN and MOBCAL sequences and the results are shown in Figure 5.5. A search window of size $W = 7$ was used, which corresponds to three levels in the hierarchical search method.

The results show the detector performance for the MAD, MSE, and Radon projection block comparison metrics. For both sequences, the MAD provided the best performance and the projection method was worst. These comparisons were

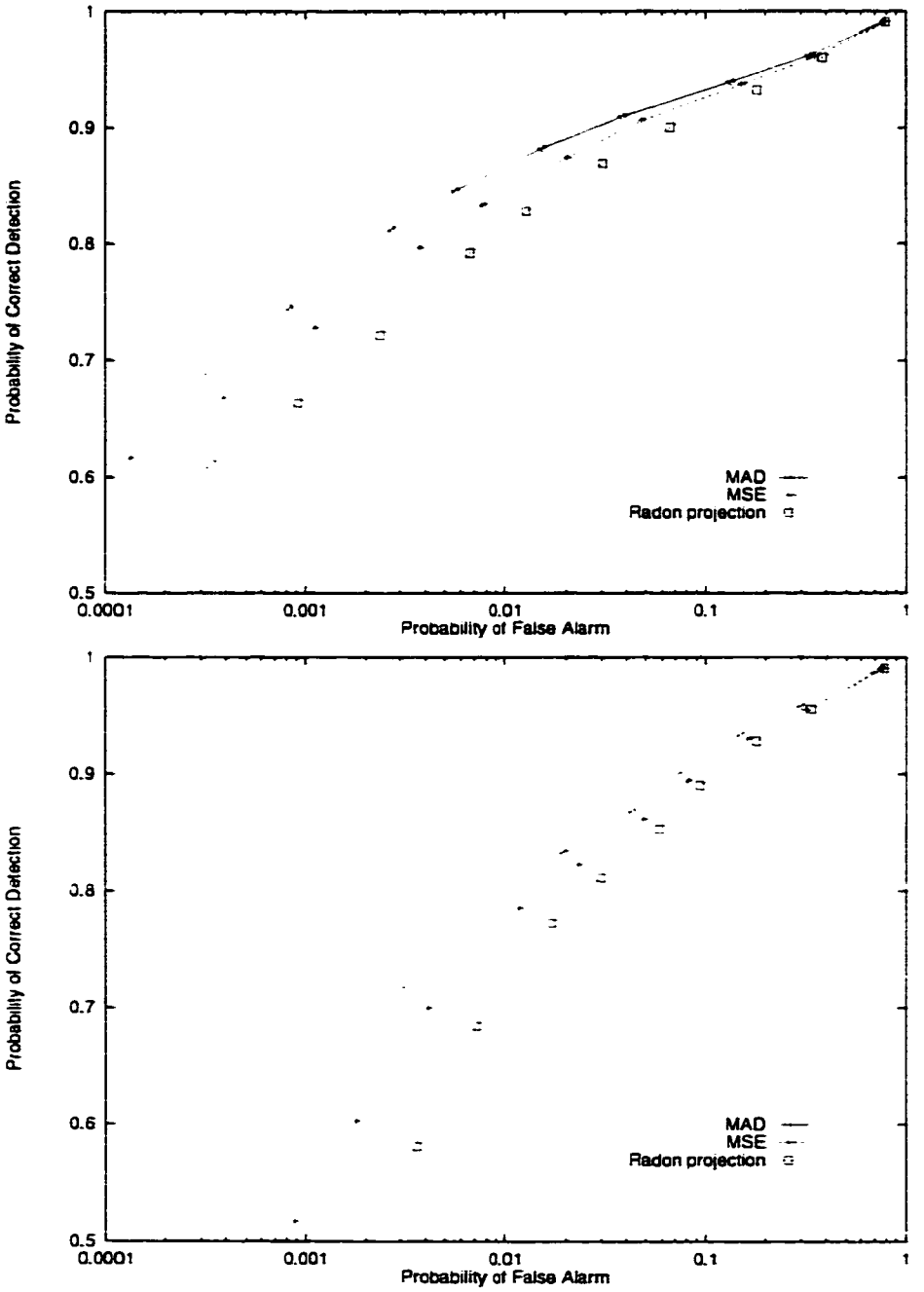


Figure 5.5: Evaluation of block comparison metrics for the WESTERN sequence (top) and MOBCAL (bottom).

verified using the exhaustive search method, and the results (not shown) were very similar.

Both MAD and MSE require $3L^2$ operations to calculate an $L \times L$ block comparison, while Radon projection requires $9L$ (except the first block). The multiple arithmetic capabilities of the C80 parallel processors makes it possible to calculate the MAD of four pixels in just two cycles (0.5 cycles/pixel), while the MSE requires one cycle per pixel [37], so the MAD calculation is twice as fast as MSE. Radon projections are potentially faster still, but the calculation of row and column sums is not well suited to PP multiple arithmetic. With the small windows used at each level for hierarchical motion estimation (typically $w = 1$), MAD requires 1728 operations versus 880 for Radon (assuming $L = 8$), using the formulas in Section 2.2. The computation advantage from Radon projection is almost two times, but without multiple arithmetic it is not likely to be faster than MAD at this window size. The MAD was therefore selected as the best block comparison metric.

5.2.3 Block Size and Search Space

The best block size for the hierarchical search algorithm was determined to be $L = 8$, and this value was used in the comparisons in Section 5.2.1. A variety of other block sizes were tried, ranging from 4 to 32. The difference in performance was fairly small within this range, but a drop in performance was seen for $L = 32$ and $L = 4$, as blocks were too large to accurately estimate motion or too susceptible to noise, respectively. Performance for block sizes of 8 and 16 was fairly similar, and 8 was selected to enable a finer-grained motion field. A smaller block size is also more convenient for a C80 implementation because more blocks can fit into on-chip memory.

More important than the block size is the extent of the search window. With the hierarchical method, this can be changed in two ways: by the number of resolution levels, and by the size of the window searched at each level. The extent of the window at full resolution is $W = w(2^N - 1)$, where w is the window searched at each level and N is the number of levels. Computation also depends on these parameters by Equation 2.14. Table 5.2 shows the full-resolution search window W and computational requirements C (in comparisons per block) for several values of w and N. It is apparent that low values of w and high values of N provide the most

	Search Window		
Levels	$w = 1$	$w = 2$	$w = 4$
$N = 1$	$W = 1$ $C = 9$	$W = 2$ $C = 25$	$W = 4$ $C = 81$
$N = 2$	$W = 3$ $C = 11.25$	$W = 6$ $C = 31.25$	$W = 12$ $C = 101.25$
$N = 3$	$W = 7$ $C = 11.8$	$W = 14$ $C = 32.8$	$W = 28$ $C = 106.3$
$N = 4$	$W = 15$ $C = 11.95$	$W = 30$ $C = 33.2$	$W = 60$ $C = 107.6$

Table 5.2: Comparison of search window and computation for hierarchical search parameters.

efficient method of covering large windows.

From a theoretical analysis of the above formulas, it would appear that an optimal selection of parameters would be $w = 1$ and N varies depending on the desired window W. In practice, this searches only the "new" vector locations in a larger level of the hierarchy; that is, as vectors are propagated from a lower resolution level, all vectors are scaled by two and therefore even, and the "new" locations are those offsets corresponding to odd vector values. If the frame information is monotonically increasing around the optimal motion offset, this will provide the best motion estimate [25]. However, it is possible for noise or artifacts of the scaling

process to cause false motion estimates at some level. A larger window w allows the algorithm to re-search some locations at the new higher resolution, possibly accepting a location already searched at the previous level, making the algorithm more robust.

A comparison of hierarchical motion estimation on the WESTERN sequence with several different level window sizes is shown in Figure 5.6. Motion estimation used the MAD block comparison, block size 8, and levels were created by four-pixel averaging. The search window w at each level and the number of levels were selected from Table 5.2 to provide approximately the same full-resolution window W. Another plot of several values of w with the same number of levels is included for comparison.

From Figure 5.6, it is apparent that a search size $w = 2$ at each level provided the best performance, although the difference was small. When $w = 4$, the results were slightly worse, probably due to the effects of noise created by the scaling process. A minimal window $w = 1$ performed almost as well as $w = 2$ for the same full-resolution window size. The difference in computation, however, was almost 3 times. For the real-time restoration system, therefore, a window size $w = 1$ was selected for general use, and a larger window $w = 2$ can also be used if necessary for some sequences, at the cost of slower execution.

The performance of reducing the number of locations searched at each level, described in Section 2.3.4 as the adaptive block matching search method [13], was evaluated and compared with an exhaustive search at each level. Figure 5.7 shows a comparison between exhaustive search and reduced per-level search for the WESTERN and MOBCAL sequences. Three levels were searched for all windows in the WESTERN sequence, and four levels were used for MOBCAL. Level windows of

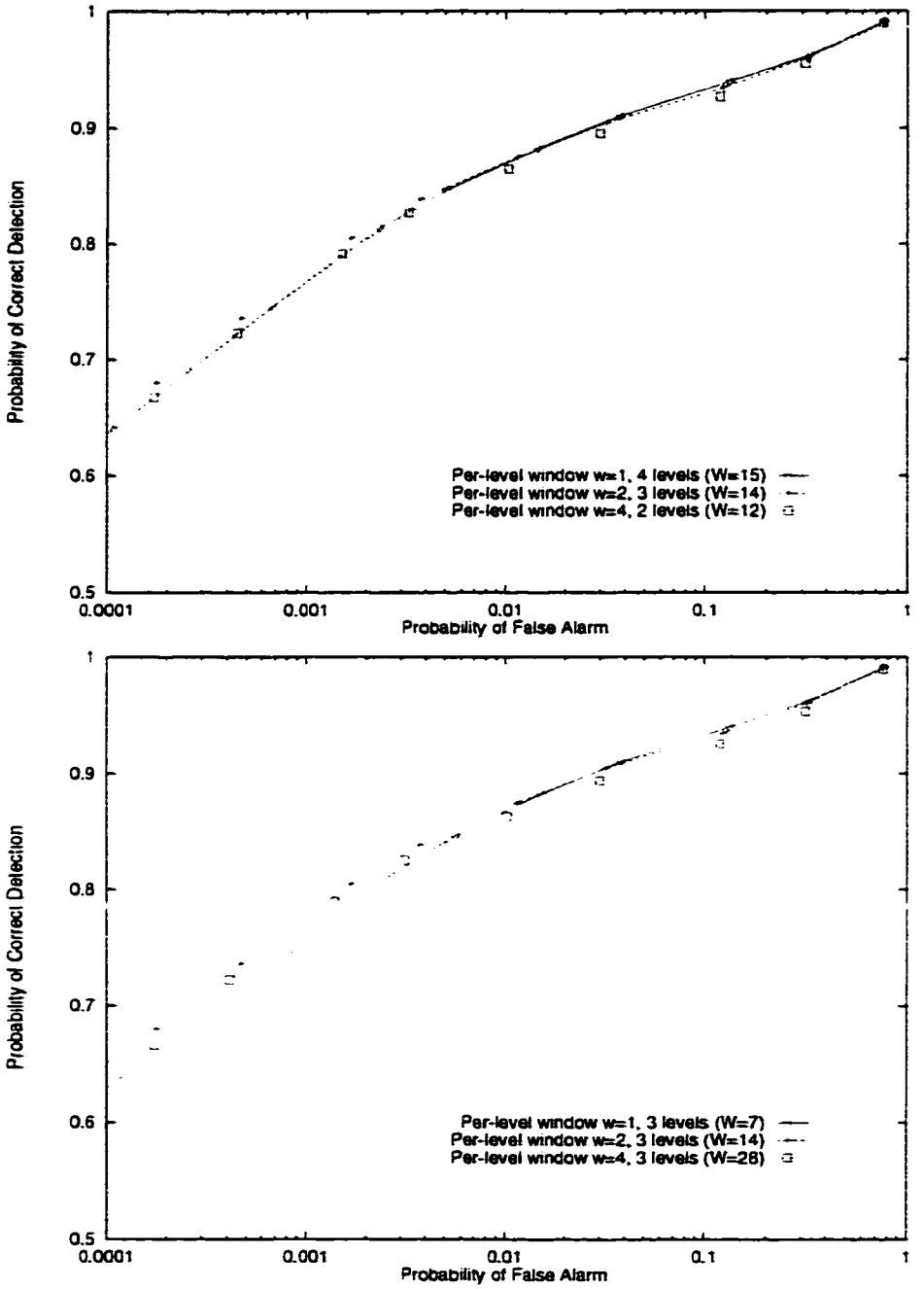


Figure 5.6: Comparison of per-level search window w and number of levels N. The number of levels was varied for an approximately equal full-resolution window W (top), and fixed at three levels (bottom).

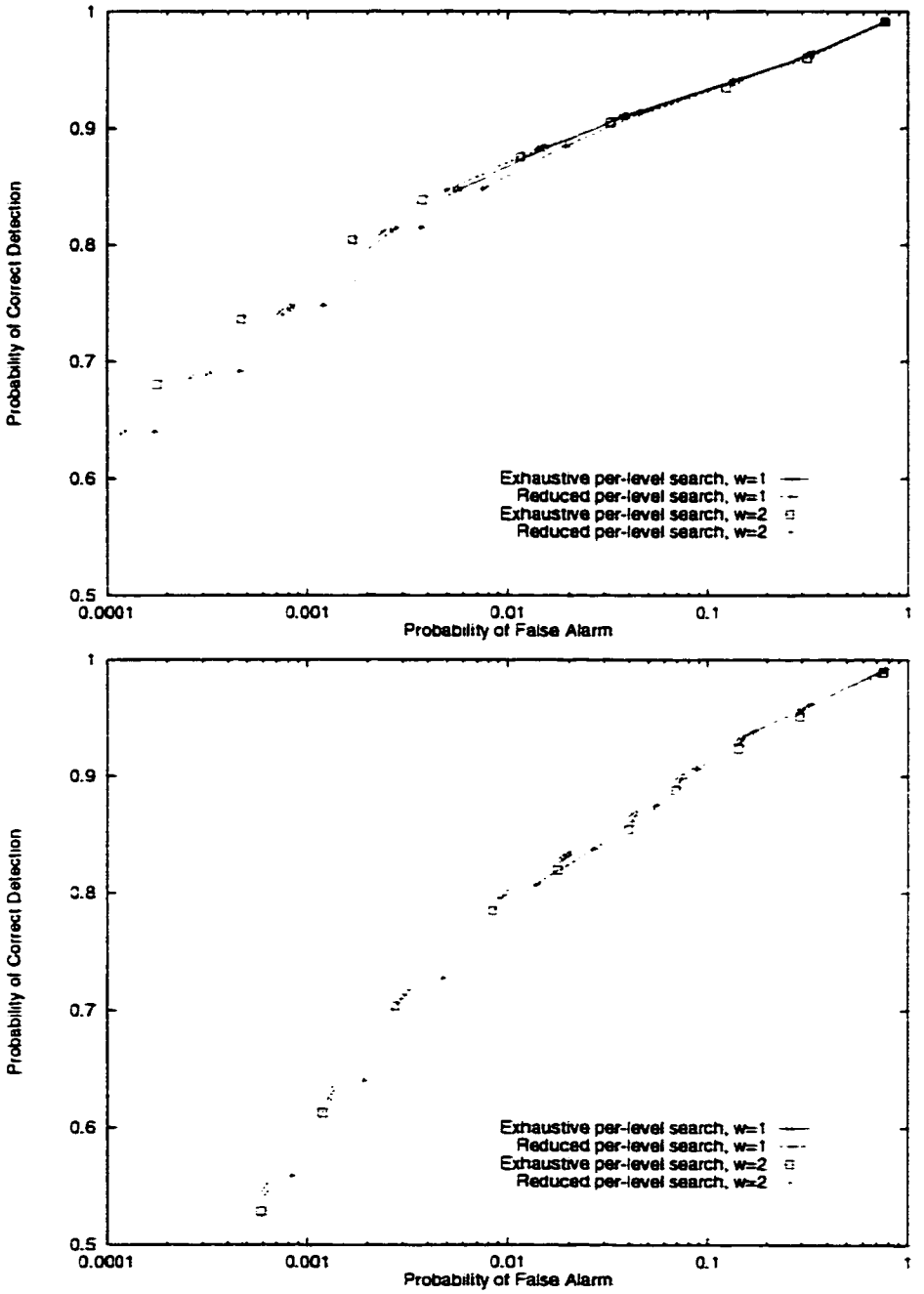


Figure 5.7: Comparison of exhaustive and reduced per-level search methods for WESTERN (top) and MOBCAL (bottom).

$w = 1$ and $w = 2$, with both exhaustive search and reduced search are shown. In general, the reduced search method performed worse than the exhaustive search but required fewer computations—9 locations versus 5 for $w = 1$ for exhaustive and reduced respectively, and 25 locations versus 8 for $w = 2$.

Using a reduced search method at each level could reduce the number of computations required, especially with $w = 2$, but the exhaustive method provided superior results. An exhaustive search with $w = 1$ was comparable to a reduced search with $w = 2$, both in performance and computation. It was found, however, that the added complexity of the adaptive block matching search algorithm made the estimation routine too large to fit in a parallel processor instruction cache block[1], and would therefore require more computation time (due to cache loads) than an exhaustive search with $w = 1$. The exhaustive search was therefore selected for both window size 1 and 2. Selecting window size 1 gives the fastest execution at acceptable accuracy, and window size 2 provides the most accuracy at a higher computational cost.

5.2.4 Image Scaling

The preceding sections have analyzed the computation required for the hierarchical search method depending on the block comparison metric and search space. These tests have included only the computation required to do the actual block search at all levels, assuming the image resolutions have already been created with four-pixel averaging. In this section, the performance of motion estimation is measured with various methods of filtering and scaling the full-resolution frame to create the image hierarchy. Filtering algorithms were previously discussed in Section 2.4.

[1] Details of how the algorithm makes use of the instruction cache is given in Section 6.1.4.

Figure 5.8 shows the results of motion estimation with four-pixel averaging, 3×3 and 9×9 Gaussian filters, and no filter, all followed by decimation by two in each dimension to create each resolution level. Both the WESTERN and MOBCAL sequences were used, with four resolution levels, a per-level window $w = 1$, and the usual settings of the MAD comparison metric with a block size of 8.

The four-pixel averaging method worked best for both sequences. This seems unintuitive when comparing the frequency response of the averaging and the 9×9 Gaussian filters in Figure 2.7: more aliasing occurs when using the averaging filter. The reason performance decreased when using the Gaussian filters was because of their greater spatial extent—damage intensities were "spread" over a larger region because they were included in the filter calculation for surrounding pixels. Increasing the damage area outweighed the effect of aliasing. The effect of using no filter, and simply decimating the images, was surprisingly good—it performed almost as well as averaging on the WESTERN sequence, and was slightly worse than the Gaussian filters on MOBCAL.

The four-pixel averaging method provided the best performance, and was also very efficient to calculate at 4 operations per decimated pixel. The Gaussian filters provided worse performance at a higher computational cost, so they were unsuitable for a real-time implementation. The decimation method with no filter provided excellent performance on the WESTERN sequence, and fair performance on MOBCAL. An advantage of this method, however, is that the decimation can be done entirely with the C80 transfer controller, which is a significant savings in computation for the entire process.

While the use of decimation without an antialiasing filter is not recommended [24], in this application it can provide good results. Most of the frequencies

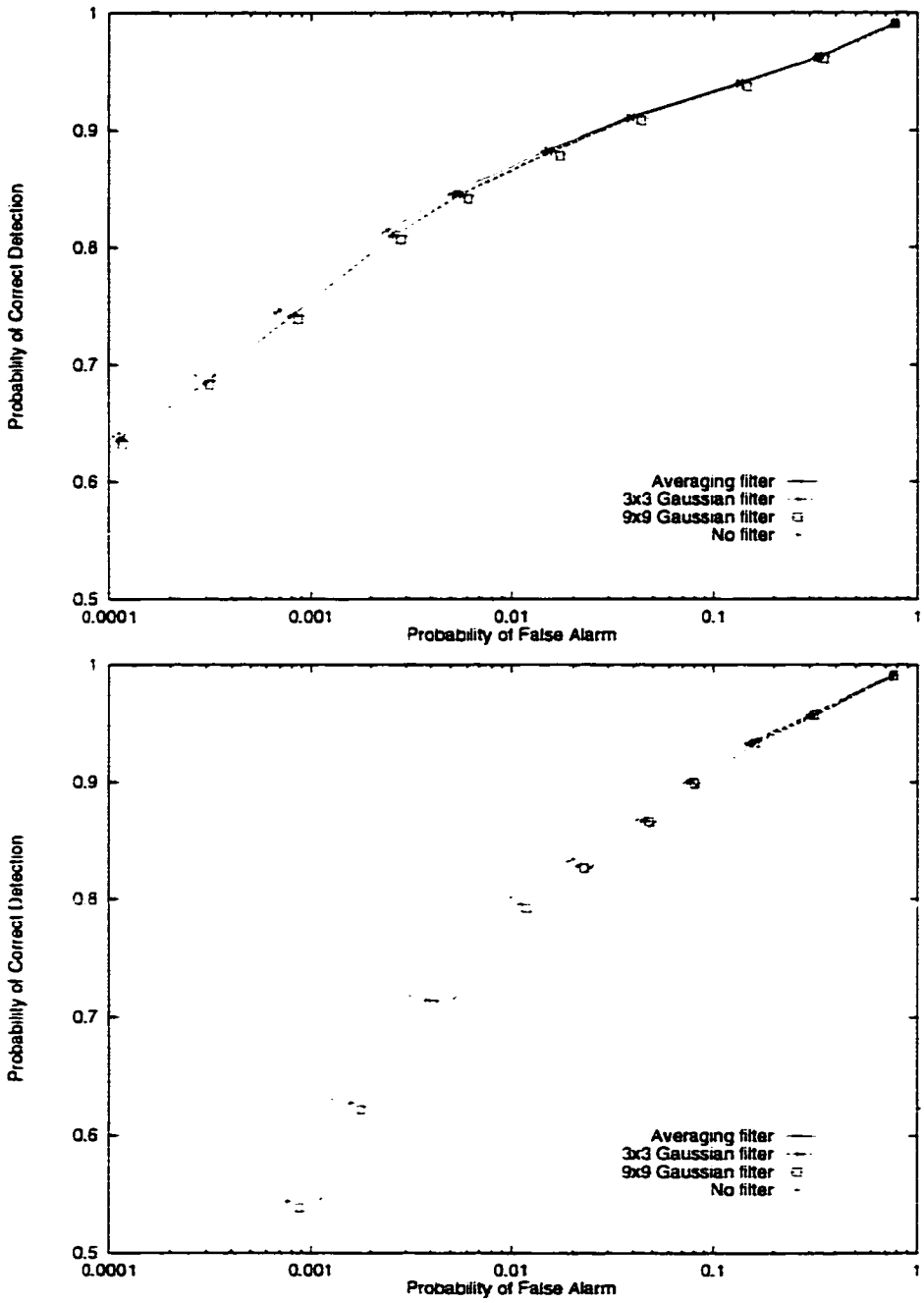


Figure 5.8: Comparison of image scaling filtering methods for WESTERN (top) and MOBCAL (bottom).

in an image are well below the Nyquist limit [28] and therefore the effects of aliasing are fairly small at the higher resolution levels. At lower resolutions, aliasing becomes significant but still does not present a serious problem—aliased information appears essentially as noise at the lower resolution, and since the motion estimator is designed to be robust to noise its performance is acceptable. Sequences containing narrow lines or small text have significant high-frequency information and therefore would alias badly, but these are fairly uncommon in motion picture films.

For the implementation of a real-time restoration system, decimation without a filter was judged to be acceptable because of the significant savings in computation. With more computing power or memory bandwidth available, however, the four-pixel averaging filter would be a useful addition.

5.2.5 Motion Vector Acceptance Thresholds

Methods for improving the accuracy of motion estimates were previously discussed in Section 2.5. The ratio of the MAD $E(\vec{v}_{n-1})$ at the best match location to the MAD $E(0)$ at zero motion is calculated as $\frac{E(\vec{v}_{n-1})}{E(0)}$. If this ratio is below a threshold t_a, the estimate is accepted: otherwise, a zero motion vector is used.

Tests at several thresholds on the WESTERN and MOBCAL sequences are shown in Figure 5.9. A plot of detector performance when no threshold was used was compared with thresholds of 1.0, 0.9, and 0.8. Ideally, a threshold of 1.0 should have the same performance as when using no threshold. However, using a threshold of 1.0 favored zero offsets when the MAD of the best and zero matches were equal, while using no threshold favored vectors in the order they were searched (left to right, top to bottom).

The performance for the WESTERN sequence showed there was a slight

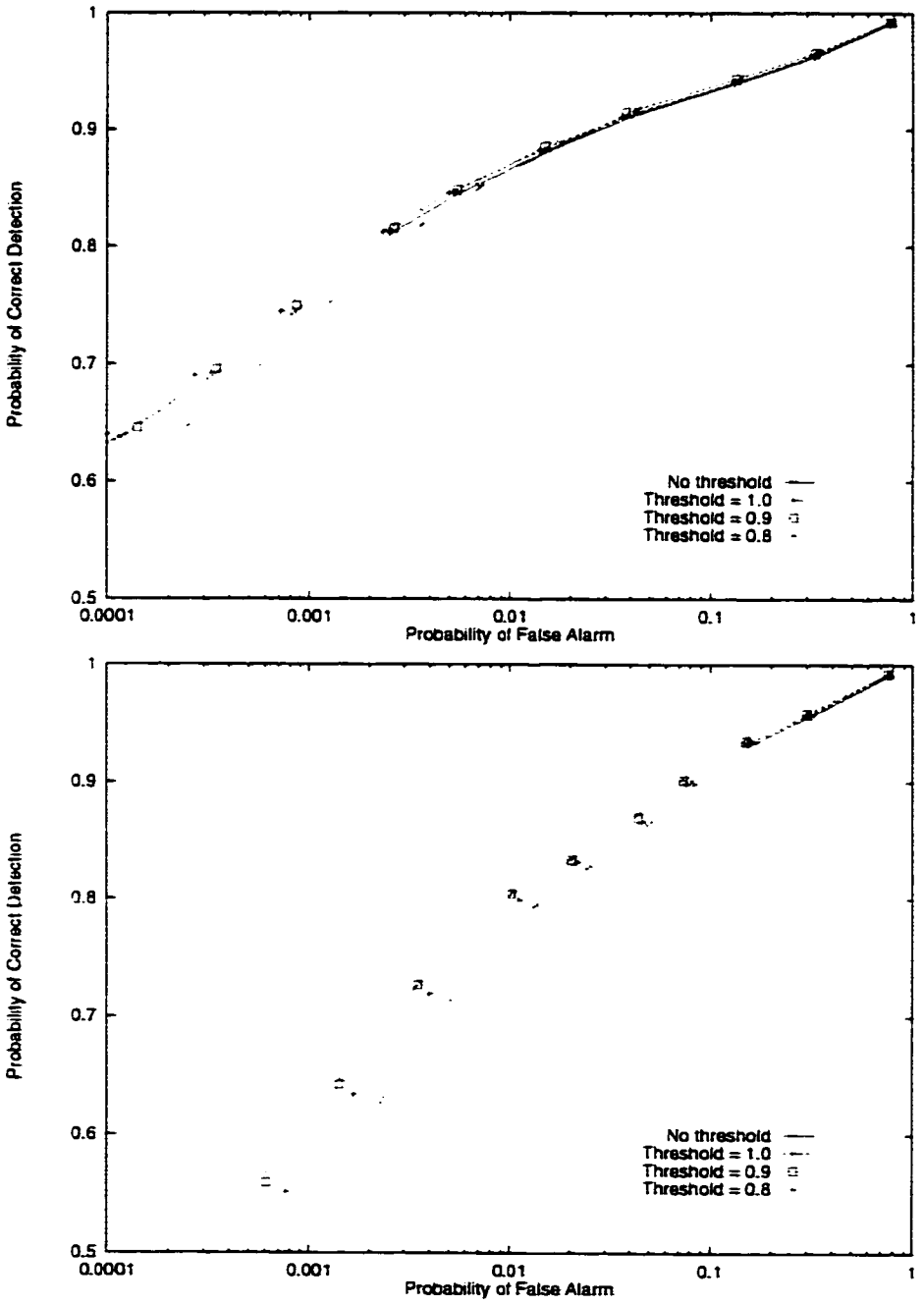


Figure 5.9: Comparison of acceptance thresholds on the WESTERN sequence (top) and MOBCAL (bottom).

improvement when a threshold of 1.0 was used, compared with no threshold, but lower values decreased performance. For the MOBCAL sequence, however, the opposite was true—performance increased significantly as thresholds were lowered toward an optimal level of approximately 0.8.

It was apparent from these tests that an acceptance threshold improved performance in some cases, but decreased performance for other sequences. The process of calculating the MAD at zero offset involves an extra block comparison per block per level. For a C80 implementation, the block data for zero offset must also be loaded into on-chip memory for calculation, requiring more memory bandwidth. It was therefore decided that using an acceptance threshold, with questionable effects on accuracy, did not justify the increase in computation and memory bandwidth.

5.2.6 Summary of Motion Estimation Methods

It was previously established that the block matching motion estimation algorithm would be the most appropriate for real-time implementation on the MX. This was due to its ability to estimate large displacements and its suitability for implementation on the C80, especially the integer parallel processors. The hierarchical block search method was most suitable, as it provided very good estimates at a low computational cost. A block size of 8×8 was determined to be a good size for search blocks. The most appropriate comparison metric used to select the best match within a search window was the mean absolute difference (MAD). This had both the best performance and the lowest computational cost.

The size of the search window w for each hierarchical level was shown to give the best performance for $w = 2$. However, this uses a large fraction of C80 computing cycles, so $w = 1$ was selected for general processing at a slight decrease

in accuracy. Both search window sizes were chosen for C80 implementation: $w = 1$ for real-time processing, and $w = 2$ for increased accuracy. The number of levels can be varied depending on the desired full-resolution search window, but a practical limit is the lowest resolution level must be larger than the search block size (8×8). An exhaustive search is used at each level, because the computational savings from using a reduced search method did not justify the increase in algorithm complexity.

Scaling the full-resolution image into levels for motion estimation is performed by decimation, without prefiltering. This method causes aliasing and motion estimation accuracy suffers slightly as a result, but eliminating the filter and using the C80 transfer controller to perform scaling was a significant savings in computation and memory bandwidth.

Using an acceptance threshold, where a motion estimate is accepted only if it is significantly better than zero motion, was shown to significantly improve performance in some sequences (MOBCAL) and offer negligible improvements for others (WESTERN). It was decided not to use an acceptance threshold because of the extra computation and memory bandwidth necessary to calculate zero offset differences.

5.3 Detection of Film Defects

A defect detection algorithm attempts to identify the locations within a frame that are most likely to be damaged. A number of methods to do this have been presented in Chapter 3. Since the aim of this work is to create a real-time restoration system, only those methods which are computationally efficient were candidates for testing. Specifically, the spike detection index (SDI, SDIa, SDIp) and rank ordered difference (ROD) detectors were chosen for this work. Other more complicated algorithms

could execute in real-time on more powerful hardware, but this implementation is based on the MX and therefore the algorithms must fit within its capabilities.

The detector tests used the motion estimation method summarized in Section 5.2.6. A window of extent $w = 1$ was searched exhaustively at four resolution levels, for a maximum motion offset of 15 pixels at full resolution. No thresholds on block acceptance were used.

Figure 5.10 shows the performance of four different detectors on the WESTERN and MOBCAL sequences. A description of this type of behavior plot was given in Section 5.2: high values of correct detection with low false alarm rates are most desirable. The SDIa and SDIp detector behaviors were plotted by varying the threshold e_t between 0 and 255. Through trial and error, it was found that the best performance for the SDI detector was obtained by keeping t_1 constant at 10 and varying t_s between 0.0 and 1.0. The ROD detector thresholds T_2 and T_3 had little effect on the performance of the detector, and were fixed at 60 and 80, respectively, while T_1 was varied from 0 to 60.

The SDIa and SDIp detectors were relatively similar in performance, with SDIp being slightly better except at very high correct detection rates. SDI was the worst performer of the detectors, and required the user to set two thresholds for best performance, making it awkward to use. ROD was the best performer overall, except for very high correct detection rates. It required three thresholds to be set, but it was relatively insensitive to T_2 and T_3 so these rarely needed to be modified.

A useful range for the detectors was between 75 and 85 percent correct detections, with the false detection rate as low as permitted by the detector performance. Low values of the detector threshold resulted in many detections, increasing both the correct and the false alarm rate. High thresholds decreased the number of de-

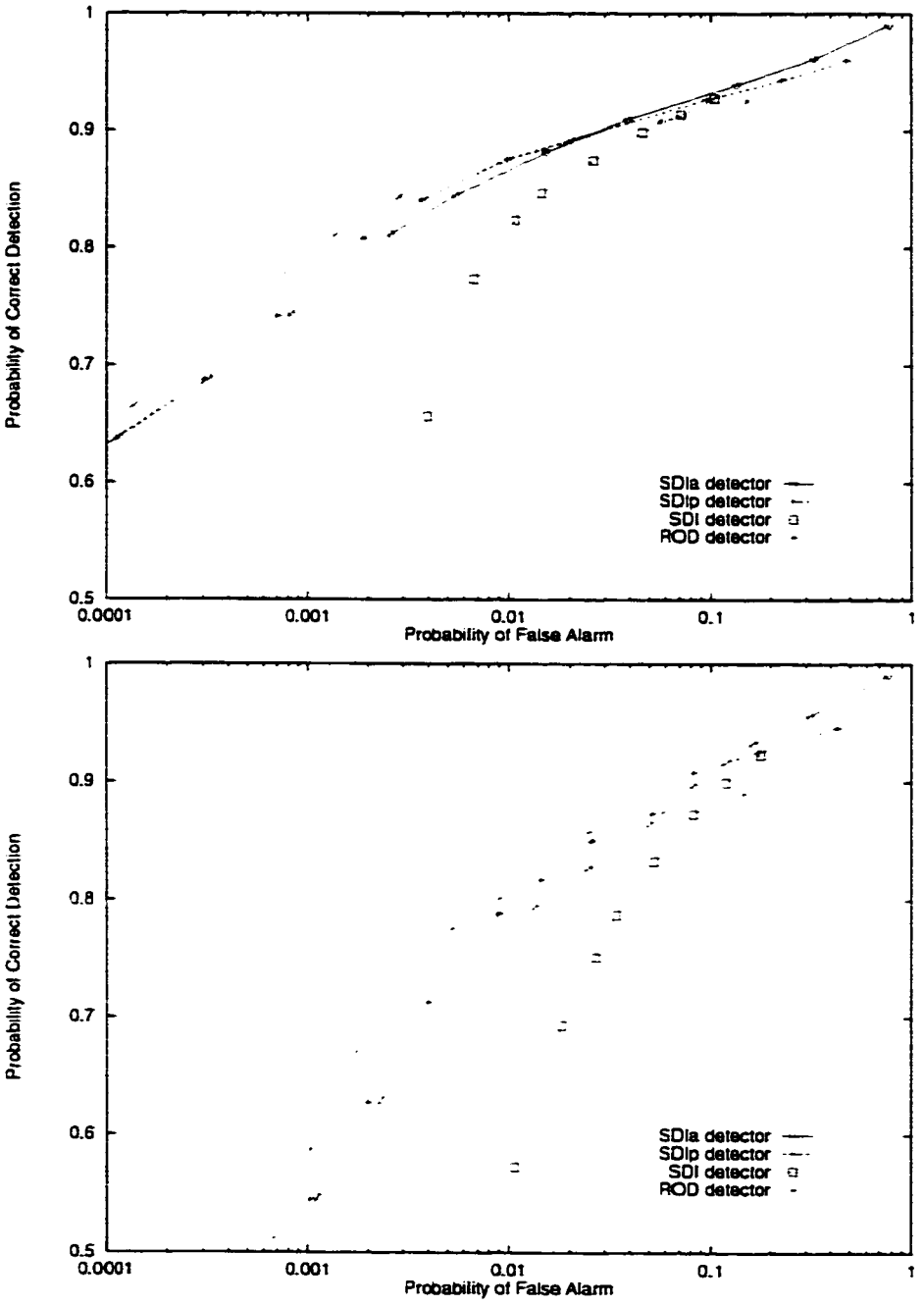
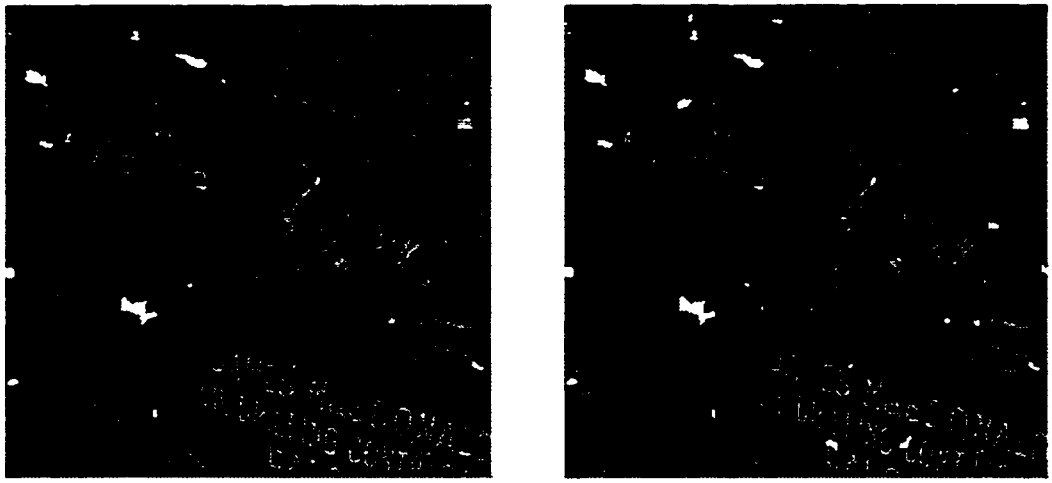


Figure 5.10: Comparison of detection algorithms on the WESTERN sequence (top) and MOBCAL (bottom).

tections and caused the opposite effect. The impact of detector threshold on correct or false detection rates was not linear—the low range (for example, below 30 for SDIa) had a much more significant impact on performance than the higher range.

Frame 8 from the WESTERN sequence is shown in Figure 5.11 with the results from several detection algorithms. The original frame (containing artificial damage) is shown, as well as a mask (overlaid on the original undamaged frame) of the actual damage locations, indicated by bright white pixels. The detector thresholds were chosen so each had approximately a 0.005 average probability of false detection. The SDIa, SDIp, ROD, and SDI detectors were tested and had average correct detection probabilities of 0.85, 0.86, 0.87, and 0.74, respectively. These results correspond to the correct detection values shown in Figure 5.10 for the WESTERN sequence at a false detection probability of 0.005. Detector threshold values are given in Figure 5.11.

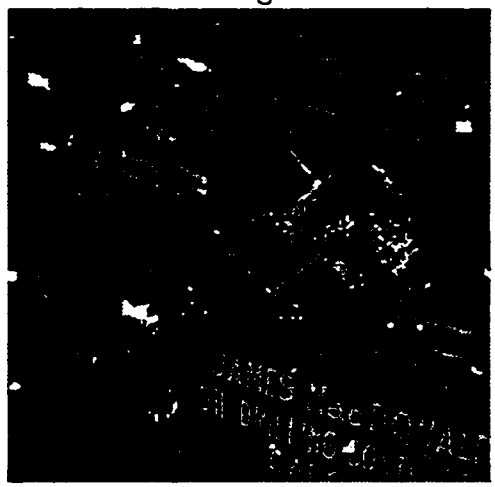
The performance of the SDIa, SDIp, and ROD detectors were similar, as indicated by their correct detection probabilities. The SDI detector performed poorly, as expected from the performance plots in Figure 5.10. The large defects below the actor's right shoulder and to the left of his head were accurately identified by all but the SDI detector. There were some differences in how the spot below the shoulder was detected—SDIa seemed to detect it the best, while ROD and SDIp missed a small bottom portion of the blotch, which could be explained by the different thresholds used in order to maintain the same false detection probability. All detectors showed some false detections around the actor's hands and the sack he is holding, which was due to poor motion estimates in this region. The ROD detector seemed more robust to these than SDIa or SDIp, and its false detections were more randomly distributed throughout the frame, as with SDI.



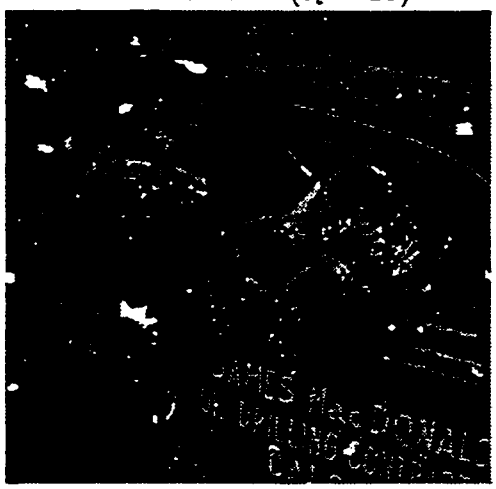
Original with simulated damage.

Actual damage locations

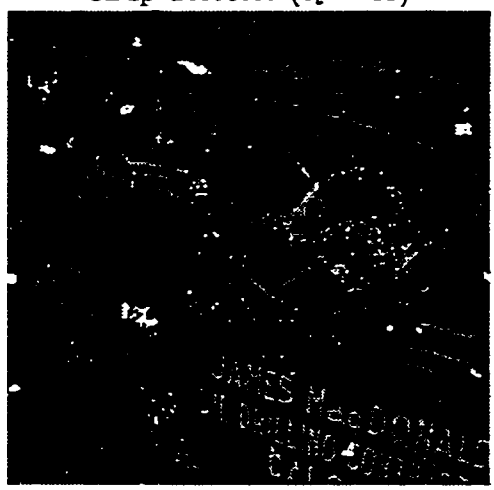
SDIa detector ($e_t = 15$)



SDIp detector ($e_t = 13$)



ROD detector ($T_1 = 7$, $T_2 = 60$, $T_3 = 80$)

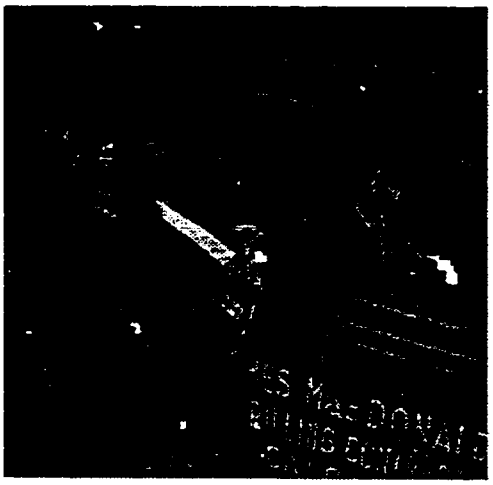


SDI detector ($t_1 = 10$, $t_s = 0.92$)

Figure 5.11: Comparison of detector performance on WESTERN frame 8. Detected damage locations are shown in white.

Note that the simulated damage in the sequence uses a random greyscale value for each damage location. This means that some defects will be more visible than others, in the case of contrasting damage and background intensity. In situations where a dark spot lies over a dark portion of the frame (or light over light), the defect is barely visible to the viewer and a detection algorithm will have difficulty identifying it. It is therefore unreasonable to assume that a detector will identify all the locations containing "actual" damage—only damage which contrasts with the background can be detected. This fact also applies to the detector plots in Figure 5.10: correct detections were measured by comparisons with actual damage locations, regardless of contrast. The effect of this on the plots was minor because all detectors were affected equally, so relative comparisons were valid.

It has been stated above that motion estimation accuracy is a very important factor in how well damage locations are identified and concealed. When objects are covered or uncovered within the scene, however, the motion estimation algorithms cannot match the missing information between frames. Figure 5.12 shows an example of this problem. Frames 45, 46, and 47 from the damaged WESTERN sequence are shown, as well as the detected damage in frame 46 using the SDIa detector ($e_t = 15$). The lapel of the actor's jacket moves from fully visible in frame 45 to fully covered in frame 47. Motion estimates for frame 46 were therefore inaccurate because the block matching algorithm could not find areas corresponding to the shape of the lapel. The detection algorithm had no valid reference information in the previous or next frames, so it identified a large area as damaged. A similar problem occurred at the bottom of the frame where an area was covered when the jacket moved downward. These types of situations occur in many scenes, and the quality of the restored output at false detection locations depends on the method



Damaged frame 45

Damaged frame 46

Damaged frame 47

Detected damage frame 46 (SDIa, $e_t = 15$)

Figure 5.12: Performance of a detection algorithm with motion estimation errors.

used to conceal damaged regions, discussed in the next section.

From the plots of detector performance and the example detection frames, it was apparent that the SDI detector was inaccurate and difficult to use because it required the user to set two thresholds. The SDIa, SDIp, and ROD detectors all were similar in performance, with the ROD detector being slightly better within the desired correct detection range of 75 to 85 percent. All three detectors were quite efficient to compute, and while the ROD detector required three thresholds, T_2 and T_3 had little impact on performance and could be kept relatively constant. The SDIa, SDIp, and ROD detectors were selected for real-time implementation, with SDIa and SDIp being the fastest algorithms, and ROD being slightly more accurate, possibly at the cost of real-time execution.

5.4 Concealment of Film Damage

The previous sections have described methods to reduce the effects of motion between consecutive frames and detect the locations of damage within a frame. Concealing the defect locations is the final step of the restoration process. A suitable concealment algorithm will use information in the previous, next, and/or current frame in order to replace the damaged pixels with values that match, as closely as possible, the original undamaged pixel data. For a real-time application, the concealment method must also be computationally efficient.

Concealment algorithms were evaluated using the WESTERN and MOBCAL sequences, and the quality of the output was determined by applying the restoration algorithms to the artificially damaged sequences, and comparing the final output with the original undamaged frames. In this way, a quantitative measure could be obtained which represents the accuracy of concealment. The accumulated mean

square error (AMSE) was used for this purpose [3]:

$$AMSE(n) = \sum_{k=1}^{n} \frac{1}{N^2} \sum_{\vec{r} \in \mathbf{N^2}} (I_k(\vec{r}) - \tilde{I}_k(\vec{r}))^2 \tag{5.2}$$

where $I_k(\vec{r})$ and $\tilde{I}_k(\vec{r})$ represent the pixel at location $\vec{r}$ in the $N \times N$ original and restored frames, respectively, and $\mathbf{N^2}$ represents all locations within the frame. Calculating $AMSE(n)$ involves computing the mean square error between the original and restored frames, and summing these for all frames up to n (inclusive). Note that the first frame (frame 0) of the sequence is not used in the calculation because it lacks a previous frame for reference, and therefore cannot be properly restored.

Since the original WESTERN and MOBCAL sequences are available, it is possible to exactly determine the defect locations and apply the concealment algorithm only to these locations. Such a strategy would evaluate only concealment without the effects of false or missed detections. However, as seen in the previous section, the detector always marked some false defect locations due to incorrect motion estimates or frame differences not due to damage. It is important for any concealment algorithm to be able to deal with these situations, and restore the false detection locations with values close to their original intensity. Therefore, it was decided to evaluate concealment algorithms using defect locations marked by a detector, chosen to be SDIa with a threshold $e_t = 15$. This tested the concealment algorithms in a "real world" situation. Motion estimation used the same technique described in Section 5.2.6, with four levels and a search window $w = 1$ at each level, for a full-resolution search extent of 15 pixels.

Figure 5.13 shows the AMSE plots for the WESTERN sequence for a variety of concealment algorithms. Averaging filters with 1:0:1, 9:0:9, and 9:8:9 support were evaluated, and the larger filters were used with and without considering damaged pixels in the support region, as discussed in Section 4.1. The median filters with

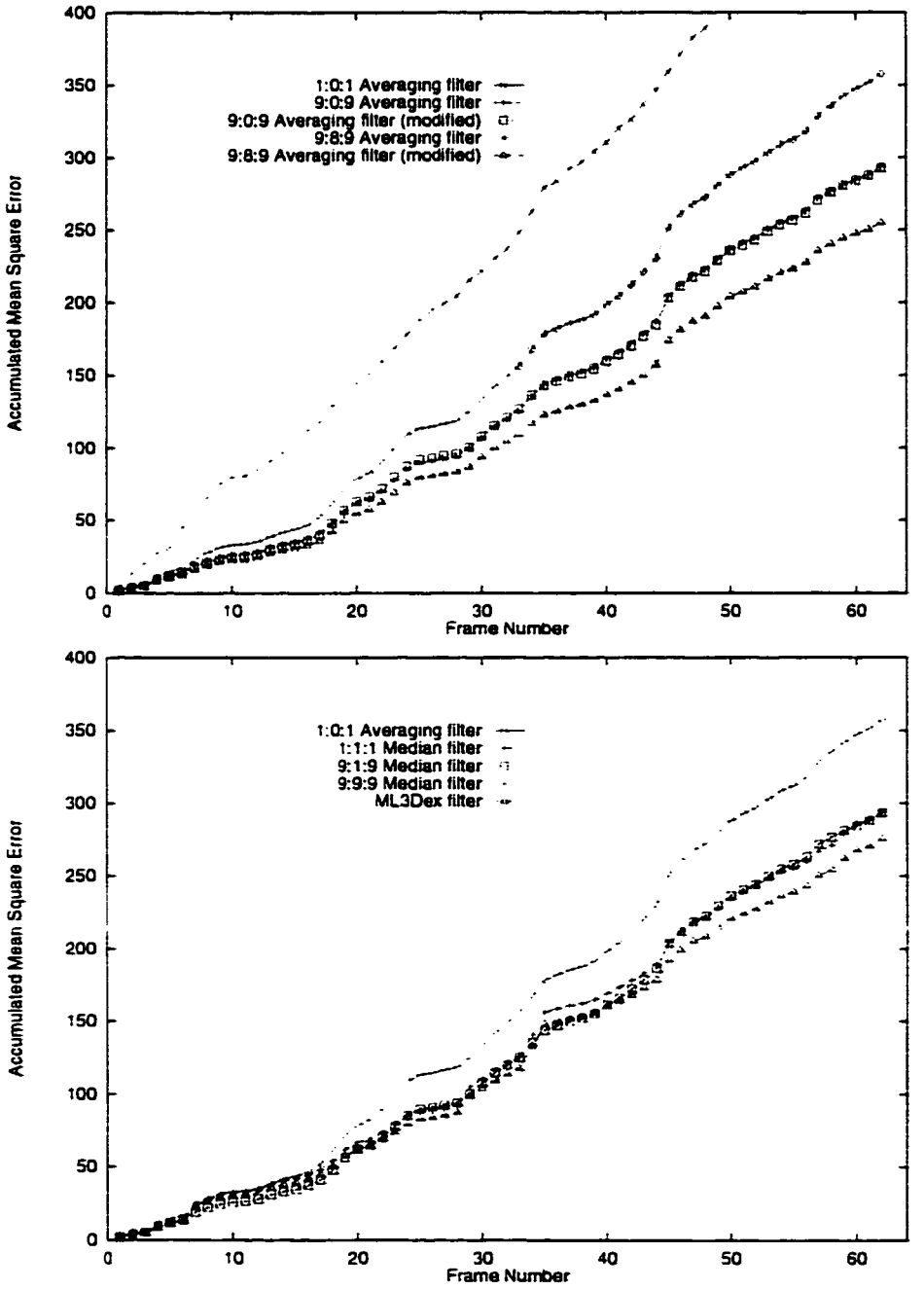


Figure 5.13: Performance of concealment algorithms on the WESTERN sequence. Averaging filters are shown in the top plot, median filters below with the 1:0:1 averaging filter repeated for comparison. "Modified" versions of the averaging filters excluded damaged pixels from their support region.

1:1:1, 9:1:9, and 9:9:9 support, as well as the ML3Dex filter, were also tested and are shown in the lower plot of Figure 5.13 with the averaging 1:0:1 output repeated for comparison. The relative performance of the algorithms on the MOBCAL sequence was similar, so only the results for WESTERN are shown.

Considering the averaging filters, it can be seen that larger support regions increased performance when damaged pixels were not included in the average calculation. Compare the performance of 9:8:9 support with and without this enhancement—damaged pixels within the current frame are often adjacent, so they adversely affect the restored output. Using only temporal support, the 9:0:9 averaging filter had approximately the same performance with or without using damaged pixels, because it was relatively unlikely that the same location would be damaged in consecutive frames, so the support region was almost always free of defects. The performance of the 1:0:1 averaging operation was worse than the others, due to its lack of spatial support.

The performance of most median filters, including 1:1:1, 9:1:9, and 9:9:9 support, was very similar and approximately equal to the 9:0:9 averaging operation. The effects of damaged pixels within the support region can be ignored, because the median operation discards these as outliers. The ML3Dex median filter had better performance, because it made more intelligent use of temporal and spatial information when choosing the best replacement pixel value.

The best overall performer was the averaging filter with 9:8:9 support, ignoring damaged pixels within the support region. However, saving and using the detected damage locations from the previously processed frame is costly both in computation and memory bandwidth. It is desirable to use techniques which are robust to damage within the support region, such as median filters. The ML3Dex

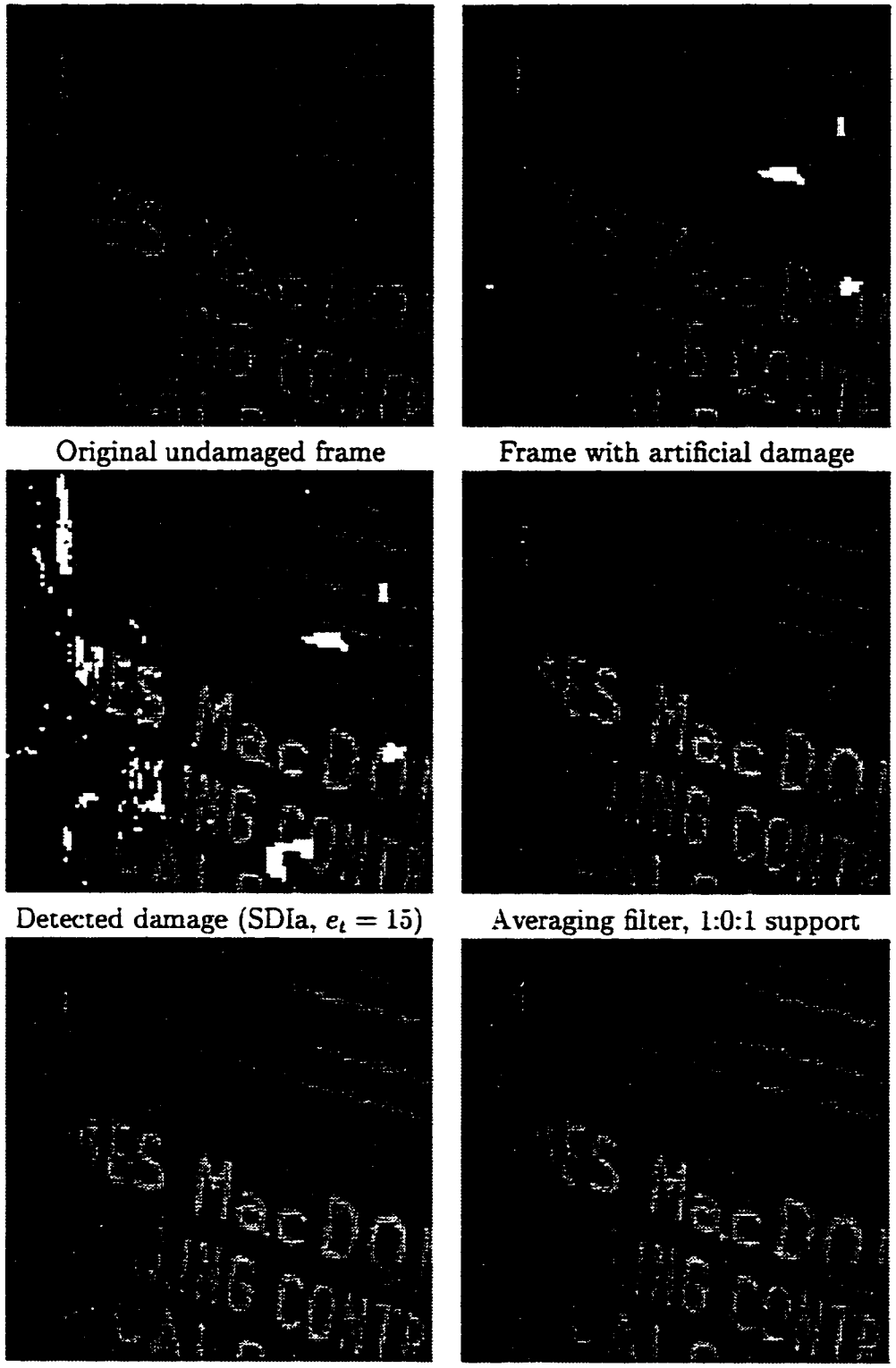
filter performed very well, and the median 1:1:1 operator also worked well at a much lower computational cost.

Computing the ML3Dex filter is too slow for real-time restoration, so the median 1:1:1 filter was chosen as the best concealment algorithm. The averaging 1:0:1 filter was also selected for further investigation because of its extremely low computational cost. Only these two concealment algorithms (median 1:1:1 and averaging 1:0:1) were considered for further testing, with ML3Dex included for comparison (although it is unsuitable for real-time implementation).

The performance of these concealment algorithms was tested on the WESTERN sequence and a portion of an example frame is shown in Figure 5.14. The original, undamaged frame as well as the frame with artificial damage are shown. The SDIa detector ($e_t = 15$) was used to identify damage locations, which are indicated as bright white pixels. The output of the averaging 1:0:1, median 1:1:1, and ML3Dex filters are shown by restoring the detected damage locations. The detected damage includes the correct defects as well as a number of false alarms, and a missed defect on the actor's hand.

In general, the defects were concealed well by all algorithms. The letters "CO", which were partly covered by a defect, were slightly blurred by the ML3Dex filter due to its use of spatial support. Median filters performed better on the false detection on the letter "M" near the actor's arm: the averaging operation distorted this region. None of the concealment algorithms were able to handle the large false detection on the jacket in the upper-left portion of the frame, because motion estimates in this region were inaccurate due to occluded areas. This light area was mostly covered in the restored frames.

Despite the problems shown in the figure, the concealment algorithms did a



Original undamaged frame | Frame with artificial damage

Detected damage (SDIa, $e_t = 15$) | Averaging filter, 1:0:1 support

Median filter, 1:1:1 support | ML3Dex filter

Figure 5.14: Concealment performance on frame 58 of WESTERN.

fairly good job of restoring the damaged sequence. Areas which had the most errors were portions of the frame which experienced fast or irregular motion. The viewer is less sensitive to damage in these regions because of the rapid motion, so errors in reconstruction go mostly unnoticed.

The best algorithm for concealment, considering both accuracy and computational cost, was the median filter with 1:1:1 support which was selected for real-time implementation. An averaging filter with 1:0:1 support was also selected for implementation, because it requires very low computation and is useful for comparison.

5.5 Recursive Temporal Filtering

All of the algorithms up to this point have used three frames from the original damaged sequence for processing. An alternative is to use the restored current frame as the "previous" frame when the next frame is processed, and so on: the current and "next" frame, obviously, have to be from the original sequence. The intent behind this feedback method is the restored frame is likely more similar to the original than the damaged frame, and therefore will allow more accurate motion estimation, detection, and concealment.

Figure 5.15 shows the AMSE method on the WESTERN sequence, with and without using feedback. A SDIa detector with $e_t = 15$ and a 1:1:1 median filter was used. Restored frames were used as the "previous" frame for motion estimation, detection, and concealment. The detector performance was virtually identical between the two methods.

From Figure 5.15, there was a slight improvement when using feedback, but the effect was small. This was mostly due to the fact that damaged locations within one frame were unlikely to also be damaged in the previous or next frames. The

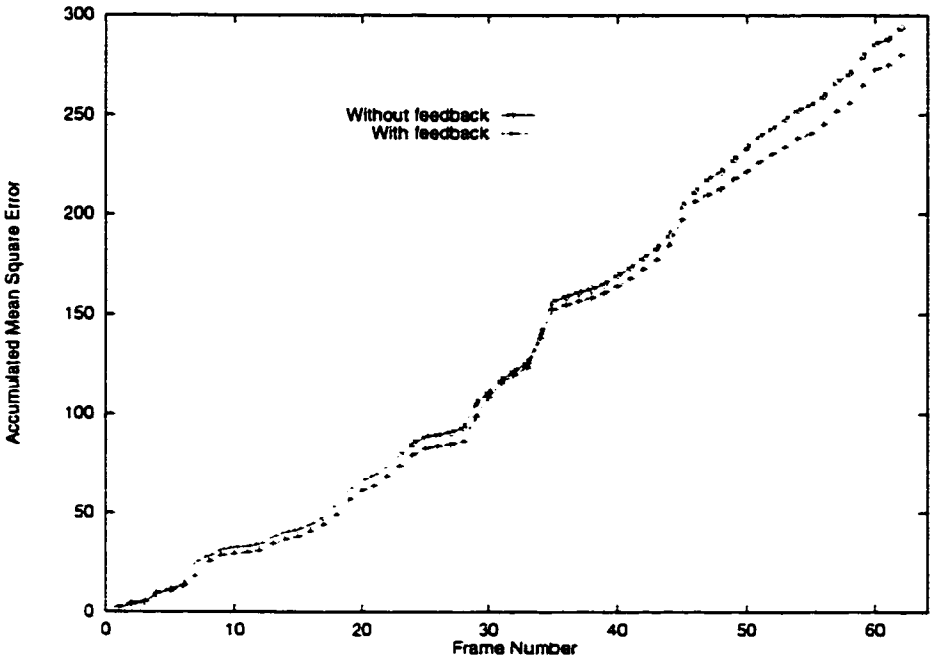


Figure 5.15: Comparison of concealment performance with and without using feedback (SDIa $e_t = 15$, 1:1:1 median filter).

restoration process, therefore, usually had an uncorrupted reference region in the surrounding frames, so feedback would not significantly improve performance. In fact, it is possible that incorrectly restored areas will propagate through several consecutive frames by employing feedback, because they cause motion estimation and detection errors. As seen from Figure 5.15, around frame 30 there was a sharp increase in AMSE for both methods, but the feedback algorithm also had a higher error in the next frame due to the errors in the restored previous frame propagating to the next.

Adding feedback to the restoration algorithm requires no extra computation, because the system needs only to change its source of "previous" frame data from the original frame to the restored frame. Feedback was selected as an option for the real-time implementation, although it offers minimal improvement.

5.6 Summary of Selected Algorithms

In this chapter, a variety of methods for motion estimation, defect detection, and damage concealment have been evaluated based on accuracy and computational cost. Only those algorithms which are suitable for real-time implementation on the MX were considered for the proposed restoration system. Evaluation was based on the artificially damaged WESTERN and MOBCAL sequences, which show a wide variety of motion and frame content and therefore provide a good indication of restoration performance on other possible sequences.

A hierarchical motion estimation algorithm was selected for its robustness to noise and its computational efficiency. A block size of 8×8 was chosen to be the best, and the MAD block comparison metric was the most accurate. Up to four image levels, including full resolution, were created by simply decimating the previous level by a factor of two; although this caused aliasing, the computational savings were significant enough to justify the reduction in accuracy. A maximum search extent of 1 or 2 pixels were exhaustively searched at each level, for a total search extent of 15 or 30 pixels at full resolution.

The spike detection index detectors SDIa and SDIp, as well as the rank ordered difference (ROD) detector gave good detection performance and computational efficiency. The detection threshold varies from 0 (high correct and false alarm rates) to 255 for SDIa and SDIp, and can be controlled by the user to select the tradeoff between correct and false detections. The ROD detector requires three thresholds, but the upper two have little impact on detector performance; values of $T_2 = 60$ and $T_3 = 80$ were found to be acceptable. The other threshold varies between 0 and T_2.

Two concealment algorithms were selected for implementation: a median

filter with 1:1:1 support, and an averaging filter with 1:0:1 support. The median filter had the best results, and the averaging operation was included for comparison because it required very little computation. Feedback, where the restored output was used as input when processing the next frame, was also an option for the real-time system.

Chapter 6

Real-Time Implementation and Performance

In the preceding chapter, various algorithms for motion estimation, defect detection, and damage concealment were compared on the basis of performance and computational cost. Those algorithms which offered acceptable performance and which could execute quickly enough for real-time processing were implemented.

This section describes in detail the software implementation of these algorithms on the Precision MX system, the user interface to the real-time restoration system, and finally some results from using the system to restore real damaged sequences.

6.1 Software

The restoration software was written for the C80 DSP on the Precision MX. It executes independently of the host PC, although the PC can send commands to change settings if necessary. A real-time restoration system could therefore be implemented,

using the existing code, on a standalone C80 processor with additional memory and video capture/display hardware. The Precision MX provides a convenient platform for developing and debugging the software under supervision of the host PC.

Texas Instruments provides a set of software development tools for Windows NT, including C compilers and assemblers for the C80 master processor (MP) and parallel processors (PP). Code for each processor is developed separately on the host PC and all code is linked together into a single executable. The host PC can then load the program into the Precision MX through the PCI bus for execution. Data and commands can also be passed between the MX and PC during program execution.

The restoration code makes use of the Texas Instruments multitasking executive functions [40]. These allow several separate program threads to execute "simultaneously" on the master processor by setting priorities to tasks and preempting lower priority tasks if necessary. The restoration code therefore has several modules which run "in the background" for reading commands from the host PC, and handling video capture and display. Note that the multitasking executive only supervises tasks which run on the master processor; the mechanism for distributing tasks to the parallel processors is described in Section 6.1.3.

Video input is in the form of an NTSC video signal and is digitized at either 640×480 or 320×240 pixels per frame. NTSC video contains 60 interlaced fields per second, which are combined to form 30 non-interlaced frames per second [19]. At 640×480, non-interlaced frames at 30 frames per second are captured, and at 320×240 only even fields are captured (30 per second). Display from the Precision MX is in the form of a VGA video signal, which for this application is fixed at 640×480 pixels, 60 frames per second. Each restored frame is displayed twice to

equal the output frame rate. A separate external scan converter is used to convert the VGA signal to NTSC video.

To facilitate testing, the restoration software can operate in two modes, referred to as "real-time" mode and "offline" mode. Normal execution is in real-time mode, where video is captured, processed, and displayed in real-time (defined as NTSC video rates, 30 frames per second). In the offline mode, the host PC loads individual digitized frames from a test sequence to the MX where they are processed, and the output frames, motion vectors, and other statistics are sent back to the host PC. This is useful for debugging and evaluating code before it is optimized for real-time execution. The processing algorithms are identical for both modes.

The following sections describe the design of the real-time C80 restoration software.

6.1.1 Algorithms

As discussed in Section 5.6, the hierarchical motion estimation method was selected for real-time implementation because of its excellent performance, noise immunity, and low computational cost. The block size was fixed at 8×8 pixels, and the search window size can be varied from zero (no motion estimation) to 2, with a maximum of 5 hierarchical levels. The maximum search offset is therefore 62 pixels (window size 2, levels 0 to 4) which is more than adequate for most sequences. The MAD block comparison metric was used, with a full window search at each level. Subsampled images are created using unfiltered decimation.

The maximum resolution supported is 640×480 pixels, which is roughly equal to NTSC video resolution. However, the amount of data which had to be transferred for real-time motion estimation at this resolution exceeded the available bandwidth

of the MX. The solution was to estimate only to level 1 (320×240) at this resolution, and then propagate the vectors to the full 640×480 resolution. Motion estimates are accurate only to within two pixels using this scheme, but computation is reduced by almost four times. In the offline mode, full scale motion estimation is possible because processing time is not an issue.

For detection, only the SDIa, SDIp, and ROD algorithms had the computational efficiency to execute in real-time and so they were the only algorithms implemented. Their performance was equal or superior to any other detection methods (with the exception of JOMBADI), with the ROD detector offering better performance at a higher computational cost. The SDIa detector is simpler (faster) and has the advantage of needing only one user-specified threshold, versus three for ROD. Other options are available for disabling detection (and concealment as well) or marking every pixel as a defect, resulting in concealment filtering across the entire frame.

Only the simplest concealment algorithms were implemented in order to execute near real-time. Purely temporal methods were shown in Section 5.4 to provide adequate performance, and the 1:0:1 averaging operation and 1:1:1 median filters were both implemented. In addition, several other formats for frame output are possible: the detection mask can be displayed, containing white or black pixels corresponding to damaged or undamaged locations: the previous or next motion compensated frames can be shown, indicating the accuracy of the motion estimates; and finally, the unprocessed input frame can be directly displayed to disable the restoration process.

For efficiency, the detection and concealment algorithms were combined as a single function for each significant combination of detector and concealment filter.

This offers maximum performance, since blocks of data need to be loaded into C80 on-chip memory only once for combined detection and concealment. It also makes more efficient use of the PP instruction cache, as further explained in Section 6.1.4.

6.1.2 Processing Pipeline

The process of restoring a frame of video requires the previous, current, and next frames in the sequence. Because the next frame is required for processing, an extra frame of delay is necessary before the "current" frame is processed. In addition, there are extra delays while the frame is captured and prepared for processing. The real-time restoration implementation is divided into several steps for efficiency, as shown in Figure 6.1.

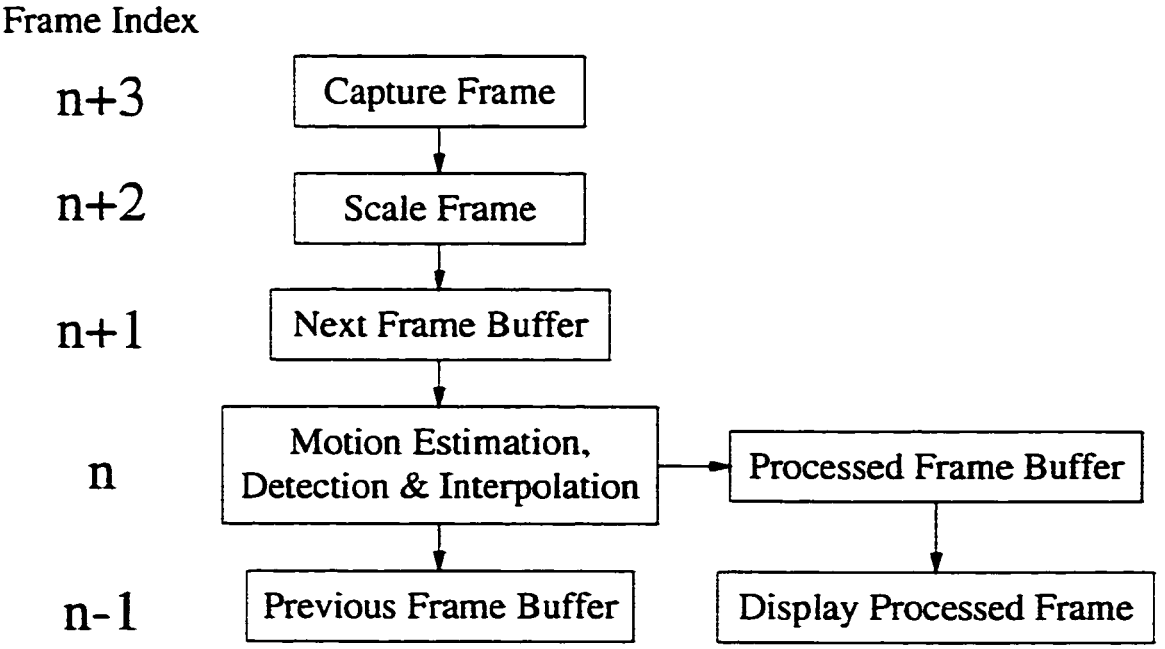


Figure 6.1: Real-time restoration system processing pipeline.

In the first step, a frame is captured into external memory using the MX capture hardware. The frame is then scaled to several resolutions using the C80

transfer controller in preparation for motion estimation (described further in Section 6.1.3). The scaled frames are then used as the "next" frame for the previous frame in the sequence. Restoration processing is performed, and finally the restored frame is displayed at the same time as the scaled frames are used as the "previous" frame when processing the next frame in the sequence.

Each step takes one frame period, and each frame of the sequence is one step further in the pipeline than the following frame. For example, while frame 10 is being processed (using frames 9 and 11 as previous and next frames, respectively), the processed frame 9 is being displayed, frame 12 is being scaled, and frame 13 is being captured. This mechanism allows the most efficient use of MX and C80 resources—capturing a frame necessarily requires a full frame period, but the C80 need only be involved once per line to transfer data from the capture FIFO to memory. Similarly, decimating frames for hierarchical motion estimation can be done with only the transfer controller, using spare memory cycles. Frame processing requires the majority of computation and memory bandwidth, and by scaling in advance more time is available for computation in this step.

Using this processing pipeline, there is a minimum delay of four frame periods between the time a frame is input and when the processed result is displayed. This delay can generally be ignored when using the system to restore video, since its effect is negligible (4 frames at 30 frames per second is 133 milliseconds). An exception is when short segments are being composited into a larger sequence—at the start of processing each short clip, there will be three "garbage" frames which should not be recorded. Audio, which is not being used in the current implementation, must be delayed by four frame periods in order to synchronize with the video output.

6.1.3 Task Distribution

The master processor (MP) uses the multitasking executive functions to supervise various aspects of the restoration system, including video capture and display, reading commands from the host PC, and distributing tasks to the parallel processors (PP). The parallel processors are responsible for all of the computation involved in the restoration process for motion estimation, detection, and concealment. The MP divides the restoration task into subtasks and allocates these dynamically to the PPs.

Commands are sent from the MP to any PP through a message queue. Parameters and identification for various tasks are placed in a linked list structure in on-chip memory by the MP. The PP polls this queue and executes the commands in order until the list is empty, signaling by a flag in the message structure when it has completed each task. The MP can monitor these queues and keep them full so the PP is never idle. Each message queue holds a maximum of two pending commands.

For hierarchical motion estimation, processing starts at the highest (lowest resolution) level image. The MP divides this image into "strips" 16 pixels high and the same width as the image, as illustrated in Figure 6.2. It places a command in a PP's message queue containing the address of the st rip and its corresponding vectors from the previous (higher) level. The PP executes this command by loading the vectors and block data for its given range, and returning the updated vectors to memory. The MP allocates all strips in the current level to PPs with empty message queue slots, waiting for an empty slot if there is none. Once all strips are allocated, it waits for all PPs to finish processing their portions before it moves to the next (lower) level. The image at this resolution is divided into strips again and the process is repeated until the lowest level has been fully processed. If the image

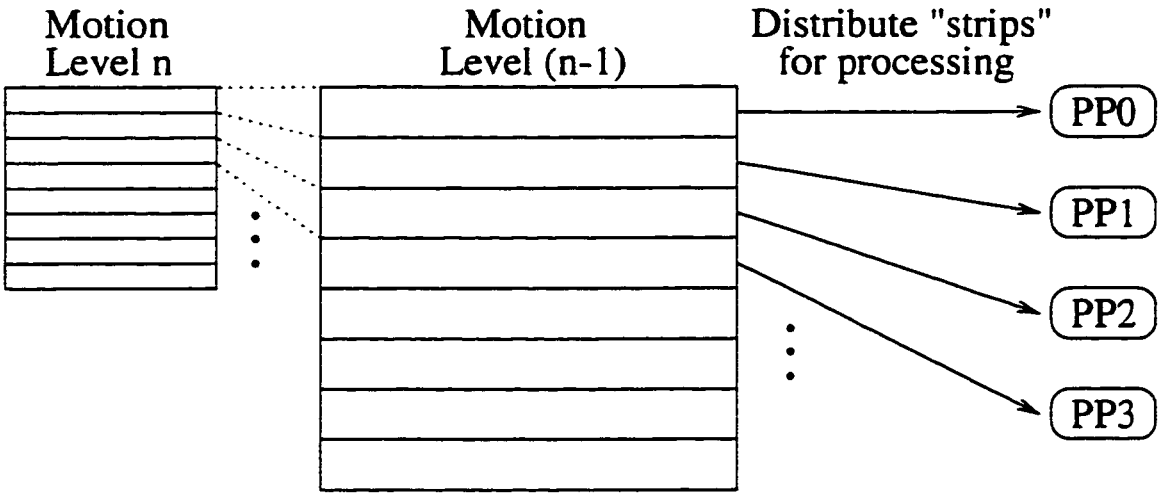


Figure 6.2: Distribution of image "strips" for PP processing.

height at any resolution is not a multiple of 16, the remaining pixels are assigned a zero motion vector and not processed: experimental tests have shown this has an insignificant effect on performance.

The end result after motion estimation is an array of motion vectors for 8×8 blocks at the lowest (highest resolution) level of the image hierarchy. The detection and concealment process then begins in a similar manner: the MP divides the full-resolution image into strips 8 pixels high[1] and allocates these to idle PPs by setting the appropriate image and vector address in its message queue. The PPs load the vectors and image data, perform the restoration operation, and transfer the processed data to the restored frame buffer. Once the MP has allocated all strips and waited for the PPs to complete their tasks, the frame has been restored.

Throughout processing, scaling of the second-previous frame (frame $n-2$ for current frame n) is taking place (refer to Figure 6.1). Before beginning the motion estimation task, the MP sets up a packet transfer structure to do the appropriate

[1] As mentioned earlier, real-time motion estimation at 640×480 only calculates vectors to level 1 (320×240). In this case, strips for detection are 16 pixels high.

scaling and starts a low-priority transfer. While the PPs are performing motion estimation and detection/concealment, any spare memory cycles are being used for the scaling process. Once all concealment tasks are completed, the MP waits for the scaling process to complete, which is usually already finished. The processed frame is displayed, the most recently captured frame is obtained, and the entire process repeats for the next frame in the sequence.

6.1.4 Block Processing

The MP divides processing of full frames into a number of "strips" which are calculated independently. Each PP, therefore, is responsible for processing a certain number of strips, as controlled by the commands the MP inserts into its message queue. The PP code is concerned only with processing a single strip at a time, regardless of the structure or size of the actual frame.

For efficiency, many parameters such as frame width and pitch, vector length, and other internal parameters which do not change each frame are precalculated. They are initialized when execution starts and each time settings (resolution, processing method, etc.) are changed by the host PC. By keeping information in PP local on-chip memory, execution time is minimized. Packet transfer structures are also kept in local memory, and only start and destination addresses need to be changed per block.

The PP code processes a number of square blocks, the sum size of which are equal to the strip length. Two separate banks of local memory are used for loading and storing data between on-chip and external memory. Figure 6.3 illustrates the process by which memory transfers and processing are pipelined. The next blocks are loaded into a memory bank while processing is being performed on the other

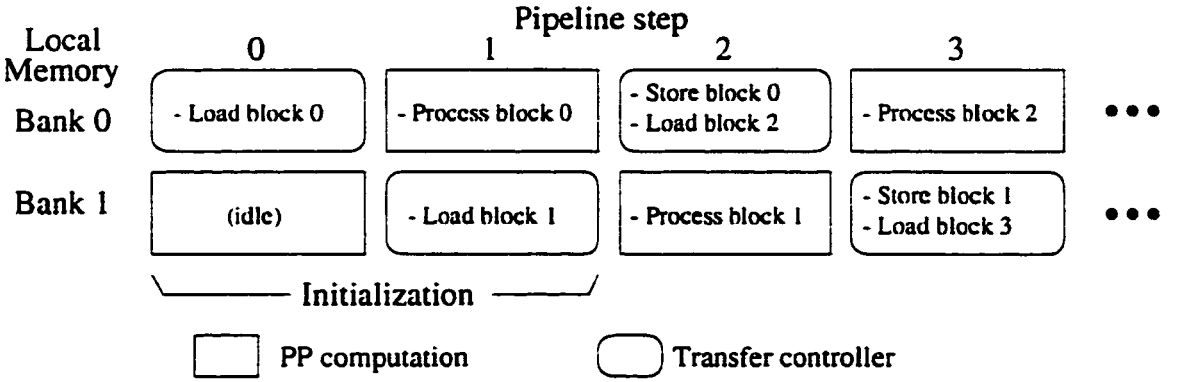


Figure 6.3: PP processing pipeline for consecutive blocks.

bank, eliminating memory conflicts within the C80 crossbar [37]. Blocks from the previous or next frame include one or two pixels of data surrounding each block for the search window. The previously processed block or vectors are transferred back out to external memory after new blocks are loaded. Memory transfers can take place concurrently with frame processing, and are set to a higher priority than the motion hierarchy scaling transfers used by the MP, so PP processing is not delayed. If memory transfers are slower than computation time for a block, or vice versa, the faster operation must wait before processing the next block. Motion estimation is typically compute-bound (computation requires more time than memory transfers), while simpler detection/concealment methods (SDIa, SDIp) are memory-bound.

Most of the advanced functions possible in the PP, such as concurrent instructions and multiple arithmetic, must be written in assembly language for maximum speed and smallest code size. All of the PP code is written in assembly language, with the exception of the "update" functions which precalculate internal variables; since parameters are rarely updated, execution time is not critical for these functions. Because of the real-time requirements of the software, it is desirable for each frame to take approximately a constant amount of time for processing. The PP code therefore makes use of conditional loads and arithmetic instructions instead of

branches to make the execution time equal for each code path. This results in the same execution time whether there are many or only a few defects in the image.

When calculating a vector offset for a block, it is important for the estimate to occur within the frame boundaries. Typically, only vectors within the frame would be considered. However, the process of checking whether each offset is in the valid range is time-consuming when conducted in the highly optimized motion estimation loop. Invalid offsets are therefore detected after all offsets for a strip have been determined: locations outside the frame boundary are reset to the estimate from the previous level. Also, because of the structure of the estimation loop, the search window is covered from lower right across to upper left. Offsets searched first are given priority if another match has the same MAD, instead of smaller offsets. Experimental tests show that neither the boundary behavior nor the search bias has a significant effect on performance.

The size and alignment of the functions are carefully controlled in order to fit efficiently into the PP instruction cache. Support code to read commands from the message queue, transfer data between internal and external memory, and other functions all fit within three cache blocks; the fourth block contains the function which does the actual processing. For example, a function to perform motion estimation within a two-pixel maximum offset is optimized to 64 instructions, exactly the size of a cache block. The instruction cache only needs to be refilled twice per frame, once for motion estimation and once for detection/concealment.

6.1.5 Control

The restoration software executes independently of the host PC, but the PC may send commands to control operating parameters, such as the motion estimation

window or the detection/concealment method. The PC can also send a number of commands which read information from the MX, such as frame rate.

Using the multitasking executive, the MP has a high-priority thread which monitors the PCI interface for commands from the PC. When a command is received, the appropriate new variables are read or written to the PC. The MP updates any necessary parameters before processing the next frame.

6.1.6 Video Capture and Display

The video capture hardware on the MX is configured when the restoration software begins execution and any time the input resolution changes. Capture is performed by a separate multitasking thread which executes each time a line of video is captured to transfer data from the capture buffer to external memory. A list of memory locations contains the addresses of several frame buffers and the capture routine fills each in order. The main restoration function reads the most recently captured frame for processing, waiting for capture to complete if necessary. For high resolutions such as 640×480, odd and even video fields are de-interlaced into the same buffer.

Display works in a similar way, where processed frame addresses are placed on a list and are displayed in order as new frames are added. Frame data must reside in video memory (VRAM).

A drawback to the capture functions is they only work in double-buffer mode, where capture currently transfers to one buffer while the other buffer is being used for processing. The rate that frames can be used from the capture hardware therefore must be a factor of the input video rate of 30 frames per second. That is, if processing takes longer than $\frac{1}{30}$ of a second, the next frame begins to overwrite the previously captured frame and processing must wait for this second frame to complete. for a

resulting frame rate of 15 frames per second.

This problem could be eliminated by buffering several frames in sequence, which would require a major rewrite of the capture software supplied with the MX. For the sake of this thesis, the frame rate problem was an acceptable limitation, since some combinations of restoration methods execute faster than 30 frames per second. The software calculates the number of cycles required for actual processing time (including memory transfers) which reflects the actual theoretical frame rate possible.

6.2 User Interface

In order to control settings of the restoration system, a graphical user interface (GUI) was developed. A screen capture of the interface, which runs under Windows NT, is shown in Figure 6.4. Selecting options will send the appropriate commands to the MX card to modify operating parameters. The results can be seen immediately in the restored output.

The GUI enables the operator to stop and start the system, change input resolution, and control system settings such as feedback. Motion estimation can be disabled or set to hierarchical search, where the number of levels and the search window can be specified. The SDIa, SDIp, or ROD detectors can be selected and their detection thresholds dynamically changed. Concealment can use a temporal averaging or median operation, as well as simply displaying the unmodified input sequence, detection mask, or the motion compensated previous or next frames. Some combinations of detector and concealment filter are meaningless, and the GUI enforces the correct combinations by disabling improper selections.

While the restoration system is running, the GUI periodically reads statistics

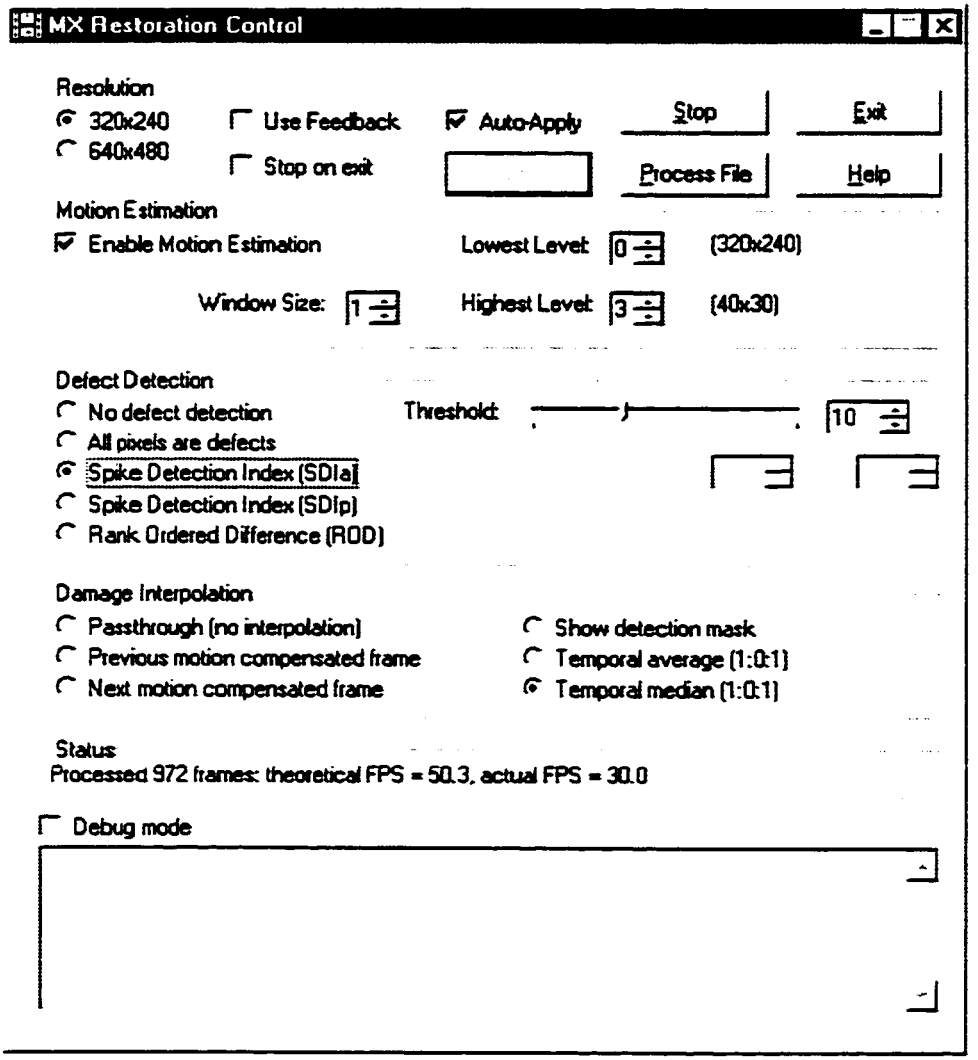


Figure 6.4: The graphical user interface for the restoration system.

from the MX containing the total number of frames processed, the actual frame rate at which the system is operating, and the theoretical frame rate calculated by the software. The actual and theoretical frame rates differ because the actual frame rate must be a multiple of the capture rate, as discussed in Section 6.1.6.

6.3 Performance

In Chapter 5, several algorithms were evaluated for motion estimation, defect detection, and damage concealment by testing them on the artificially damaged WESTERN and MOBCAL sequences. Because the exact locations of damage and the original undamaged sequences were available, a quantitative comparison of restoration performance was possible. In this chapter, the algorithms chosen for implementation in the real-time system are tested on digitized sequences from actual damaged films.

6.3.1 Restoring Real Film Damage

The algorithms implemented in the real-time system were used for restoring the FRANK and BIPLANE sequences, discussed in Section 1.4. A hierarchical block matching search was used for motion estimation, with the MAD comparison metric, four resolution levels created by unfiltered decimation, and an exhaustive search window $w = 1$ at each level for a full-resolution motion extent of 15 pixels. The SDIa, SDIp, and ROD detectors were implemented, and the averaging filter with 1:0:1 support and the 1:1:1 median filter were used for concealment.

A test frame from the FRANK sequence is shown in Figure 6.5. The original frame has some defects: the forehead, the viewer's left of the actor's head, and near

Original damaged frame

SDIa detector ($e_t = 20$)

SDIp detector ($e_t = 20$)

ROD detector ($T_1 = 15$, $T_2 = 60$, $T_3 = 80$)

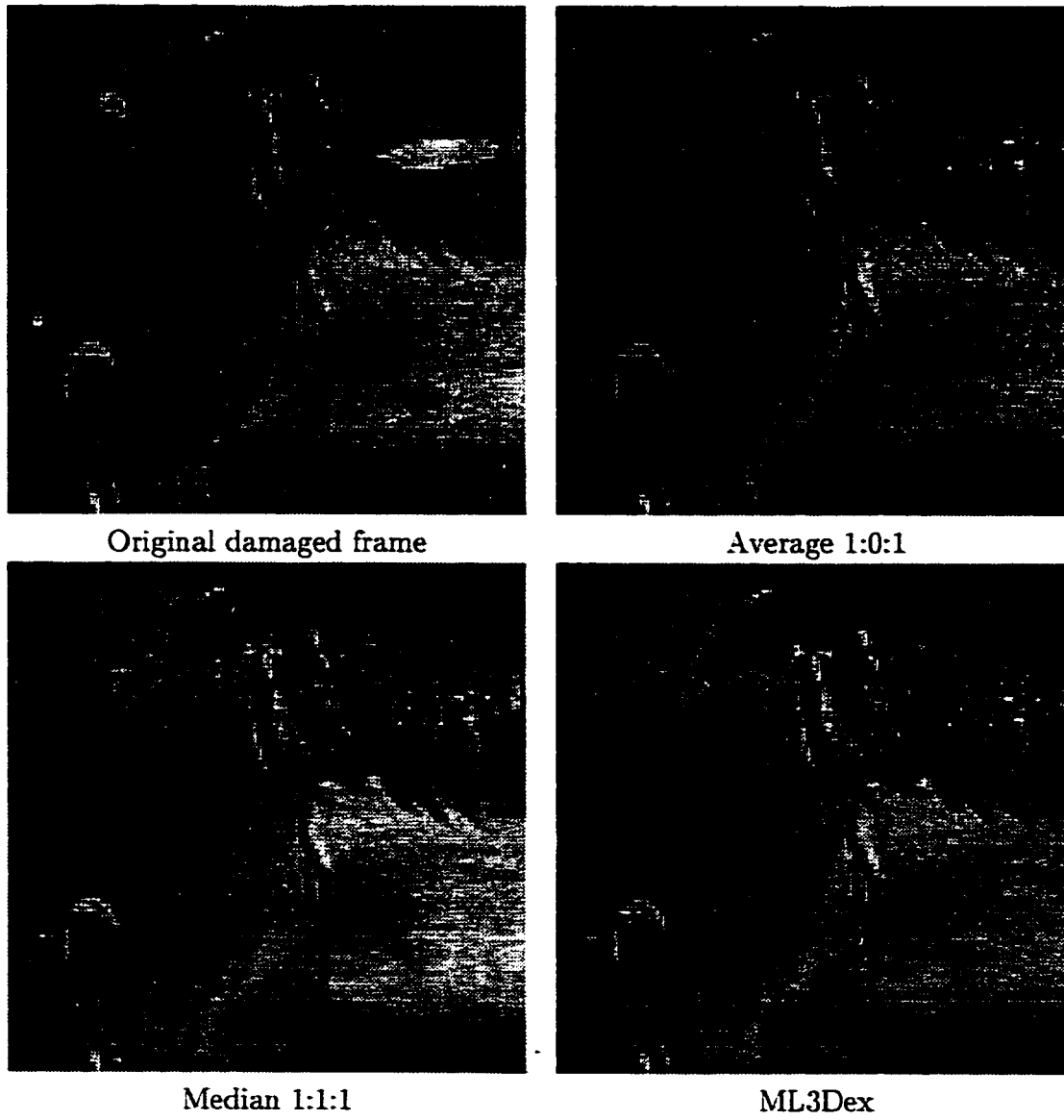
Figure 6.5: Detector performance on frame 44 of FRANK.

the shoulder on the jacket. The performance of the detectors is also shown, with detected damage shown as bright white pixels. Detector thresholds (given on figure) were chosen to accurately identify the forehead damage location with a minimum of false detections.

Motion estimates were inaccurate for the moving leaves in the background, because shapes in this region change as they move. The trees overhead cast shadows on the actor which change with the wind. All three detectors accurately identified the damaged regions despite these problems. The SDIa detection mask contained the most false detections, mostly in the moving leaves in the background and the outline of the flower. SDIp had fewer false detections because of its more strict detection criteria, and ROD performed the best because of its use of a larger support region.

A portion of the actor's head is expanded in Figure 6.6 to evaluate the performance of concealment algorithms. The original damaged frame and the restored output using the 1:0:1 averaging filter and 1:1:1 median filter are shown. Results from the ML3Dex median filter are also shown for comparison, although this method was not implemented in the real-time system. Damage locations marked by the SDIa detector ($e_t = 15$) were used; all actual damage in the displayed region was accurately identified.

Damage in the upper left of the images and just above the ear was accurately concealed, and assumed the texture of the background. The defect on the forehead was mostly covered, but the restored texture was slightly distorted compared with the rest of the hair. Differences between the concealment methods were not noticeable to the viewer in a single frame, although a slight "shadow" was apparent in this region when the sequence was replayed. The artifacts remaining from the concealment method, however, were not nearly as noticeable as the original damage.

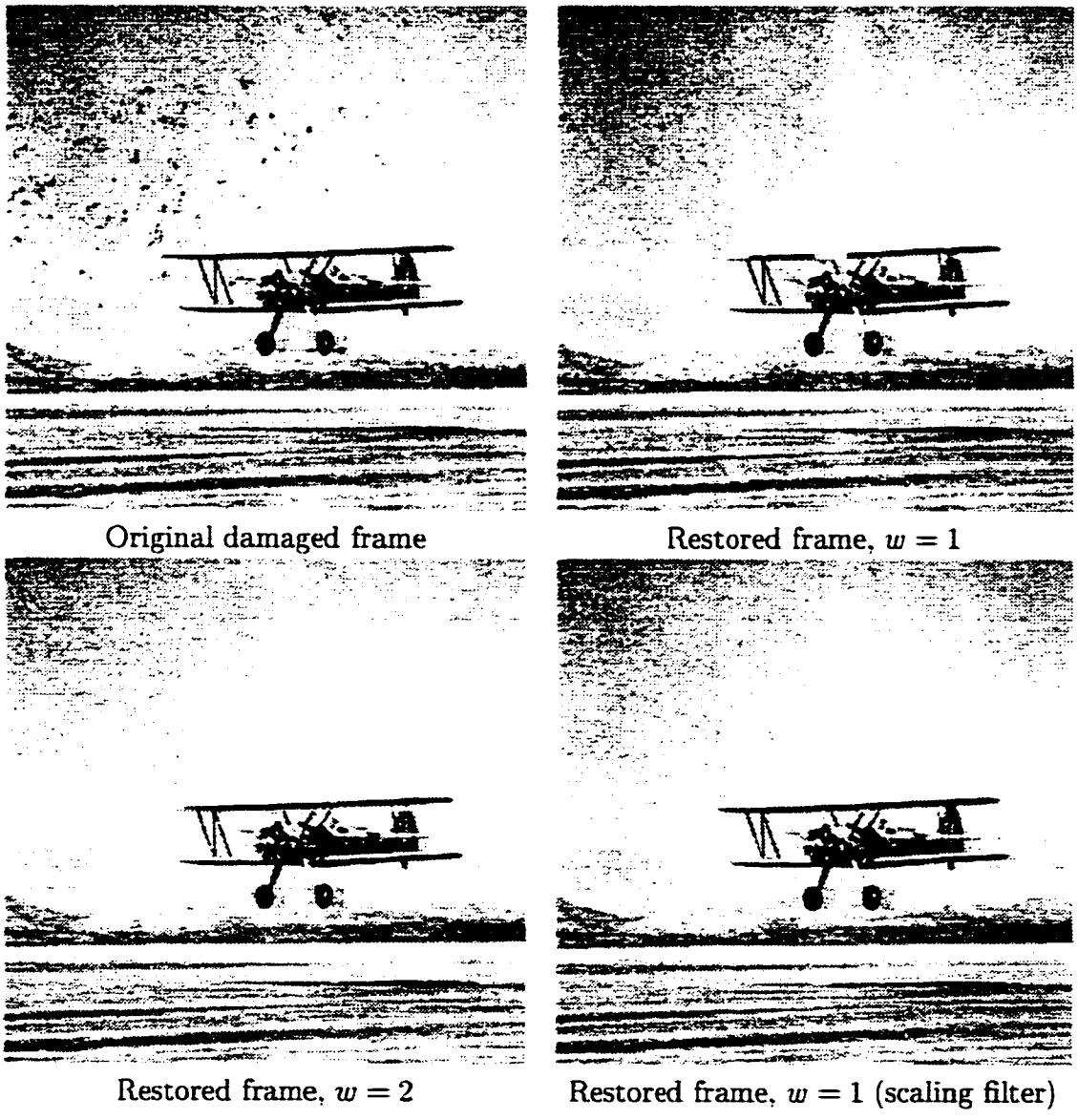


Original damaged frame | Average 1:0:1

Median 1:1:1 | ML3Dex

Figure 6.6: Concealment performance on frame 44 of FRANK.

Further testing was conducted on the BIPLANE sequence, which contains a considerable amount of damage. Figure 6.7 shows an original damaged frame as well as the restored output (SDIa $e_t = 15$, 1:1:1 median filter). Because of the increased



Original damaged frame | Restored frame, $w = 1$

Restored frame, $w = 2$ | Restored frame, $w = 1$ (scaling filter)

Figure 6.7: Restoration performance for frame 7 of BIPLANE.

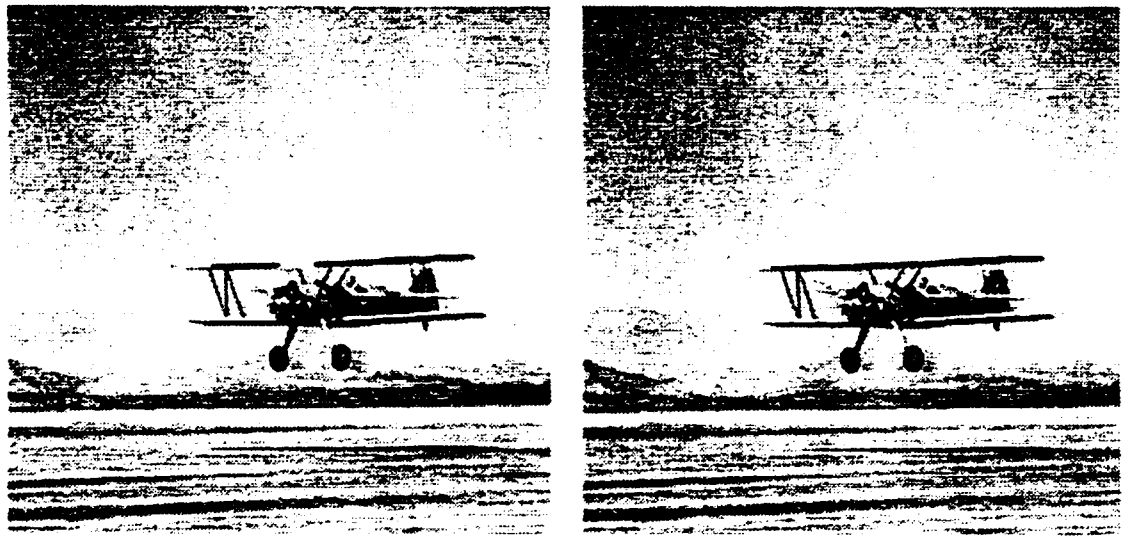
resolution of this sequence, five motion estimation levels were used, for a maximum motion offset of 31 pixels at full resolution (512×512).

Estimation with the standard per-level search window $w = 1$ was first used.

and the restoration system was able to remove the majority of damage in the frame. However, due to poor motion estimates, two sections of the top wing—the center and the wingtip—were incorrectly identified as defects and concealed with data from the background. This was mostly due to the unfiltered decimation used to create image resolutions; the narrow leading edge of the wing was eliminated at lower resolutions, and the motion estimate drifted from the proper value. By using a larger search window $w = 2$ at each level, the estimation algorithm was able to recover from poor estimates at lower levels and the center of the wing was now correctly handled, but the wingtip was still obscured. By using a four-pixel averaging filter before decimation and the original search window $w = 1$, the final image in Figure 6.7 was restored with the entire wing intact.

A further optimization in the real-time system deals with high resolution input. In order to reduce computation for large input frames (640×480), motion estimation stops at a lower resolution (320×240) in the hierarchy. This reduces computation and memory bandwidth by almost four times. Estimates are propagated to the highest level for detection and concealment, so they are accurate to within two pixels. Figure 6.8 shows the effects of this technique for per-level windows $w = 1$ and $w = 2$. Four resolution levels are used (levels 1 to 4), not including the highest resolution (level 0). Comparing the output from Figures 6.7 and 6.8, using a lower resolution for motion estimation did not have a significant impact on quality.

These results illustrate the tradeoffs made in selecting restoration algorithms for real-time execution. By using small per-level search windows and no scaling filter, motion estimation can sometimes be inaccurate, which leads to reduced restoration quality. Using lower resolutions for motion estimation with high-resolution input



Restored frame, $w = 1$ Restored frame, $w = 2$

Figure 6.8: Restored frame 7 of BIPLANE with motion estimation at reduced resolution.

makes the algorithm much faster, but the motion estimates are only accurate to within two pixels. For the BIPLANE sequence results shown in Figure 6.8, this did not affect restored quality but in some sequences with significant detail at full resolution, the output quality can be poor.

6.3.2 Real-Time Processing

The real-time system implemented on the MX was used to restore damaged film sequences which were recorded on video. Test sequences were much longer than the digitized sequences and included many scene changes, interlacing, and various frame rate conversion (pulldown) methods. The original sequences were compared with the restored output, with various settings, on the basis of visual quality.

Table 6.1 shows the theoretical processing rate, in frames per second, for various restoration settings. The theoretical processing rate is the speed at which the system could operate if it did not have to synchronize with video input; the

actual processing rate is always a factor of 30 frames per second, as discussed in Section 6.1.6. Four levels were used for motion estimation in all cases: for 320 × 240

Resolution	Search window w	Detect/Conceal	Theoretical FPS
320 × 240	(none)	SDIa/MED111	237
	(none)	ROD/MED111	60.3
	1	(none)	74.5
	1	SDIa/MED111	50.3
	1	ROD/MED111	34.4
	2	(none)	31.6
	2	SDIa/MED111	26.2
	2	ROD/MED111	21.1
640 × 480	(none)	SDIa/MED111	59.0
	(none)	ROD/MED111	15.3
	1	(none)	42.7
	1	SDIa/MED111	28.4
	1	ROD/MED111	12.9
	2	(none)	27.3
	2	SDIa/MED111	20.5
	2	ROD/MED111	10.4

Table 6.1: Real-time restoration system speed for various settings.

resolution, levels 0 to 3 were used: for 640 × 480, levels 1 to 4 were used to reduce computation, as described in the previous section. The averaging and median concealment filters have almost identical computation requirements, so only the median results are shown. SDIa and SDIp are also very similar in computing requirements, so only the SDIa results are shown.

Motion estimation with a larger search space w takes significantly more computation, as shown by the drop in frame processing rate. The ROD detector is more complex than the SDIa or SDIp detectors and therefore causes a processing rate decrease. Larger input frames decrease performance for motion estimation and detection in all cases.

As the results show, a real-time restoration system (real-time measured as at

least 30 frames per second) could use 320 × 240 resolution, with a search window $w = 1$, the SDIa or ROD detectors, and a 1:1:1 median concealment filter. More complex methods are slower than 30 frames per second. Note, however, that most can run faster than 25 frames per second, which is PAL video speed and greater than film speed. Therefore, the restoration system could run in real-time at higher resolutions, or with more complicated algorithms, with PAL video input.

If the computing requirements are too high for real-time restoration, the system will run at a reduced frame rate of 15 frames per second. This is adequate to evaluate the quality of the restored output, but the sequence is noticeably "jerky". Motion estimation accuracy also suffers due to the increased time difference between frames.

In general, most defects were removed by the real-time system. It was found during testing that some remaining defects, or artifacts from incorrect restoration, were still noticeable but the overall quality is better than the original. Scenes containing little or no motion were most easily restored. As expected from earlier simulations, sequences with large or complex motion caused the restoration algorithms to fail. These and other problems are discussed in the next section.

6.3.3 Restoration Problems

Because of the tradeoffs made in selecting algorithms for a real-time restoration system or simply because of the content in test sequences, a number of problems can occur during restoration. One example was already shown in Figure 6.7 where bad motion estimates caused parts of the image to be incorrectly concealed. The following sections describe other situations which cause restoration errors, and some possible solutions.

6.3.3.1 Motion Estimation Errors

In some scenes containing fast motion or camera pans, the motion estimation algorithms were unable to follow the motion quickly enough because of their limited search offset, and restoration quality suffered. Objects which were covered, uncovered, or changed shape were also problems for the block matching motion algorithm. Changes in brightness between frames was another source of errors.

The effects of motion estimation errors were incorrectly restored regions, which often occur at the boundaries of moving objects. These were more noticeable when the scene was mostly stationary than when they were caused by fast motion. Possible remedies are to use a more complex and computationally demanding motion estimation algorithm which is more robust. In general, however, the fact that the input sequence contains damage limits the effectiveness of any motion estimation algorithm, and so these types of errors may have to be accepted for an automatic restoration system.

6.3.3.2 Scene Changes

The restoration algorithms use the previous and next frames, as well as the current frame, in order to detect and conceal damage. On a scene change boundary, the current frame will be uncorrelated with either the previous or next frame and restoration will fail. Both the last frame of a scene and the first frame of the next scene will be affected by this problem. One test sequence contained a similar situation where photos were being taken during a press conference. Each time a flashbulb was lit, a single frame had a large intensity difference from the surrounding frames, which caused the same problems as a scene change.

For the viewer, the effects of errors at scene changes were minimal because

it required several frames for the visual system to adjust to the "shock" of a new scene. Errors in the restoration process were mostly ignored because the viewer had not yet analyzed the content of the new scene, and therefore could not easily identify defects. Artifacts in the last frame of the old scene were also somewhat eliminated by this fact—the viewer saw defects in that frame, but they were quickly "forgotten" as the new scene was studied.

There has been some work done by other researchers in the area of scene change detection [41, 42], and these methods could be applied before the restoration process. At scene change boundaries, restoration would be disabled or a modified algorithm which uses only one of the previous or next frames could be used. Another method of reducing this problem within the context of the restoration algorithm is to count the number of defects detected in each frame. If an uncommonly large amount of damage is detected, the frame can be assumed to lie on a scene boundary and the original frame can be substituted for the restored frame in display. This method would fail if the actual damage in a sequence exceeds the average error from a scene change.

6.3.3.3 Film to Video Conversion

The test sequences used for evaluating algorithms were digitized directly from film, and so did not contain artifacts from the video conversion process. When the real-time restoration system was used to restore damaged films converted to NTSC video, a number of problems were discovered that affected the quality of the restored output.

As mentioned in Section 1.3, NTSC video contains 30 interlaced frames per second, divided into even and odd fields displayed 60 per second. Depending on the

conversion process, the fields may contain information from the same or adjacent frames. In the latter case, even and odd lines are possibly uncorrelated and restoration will fail. Another problem with frame rate conversion is the necessity to repeat some film frames in the video output. This violates the assumption that the same location will not be damaged in adjacent frames and the restoration procedure will fail.

The best solution to these problems is to control the film to video conversion process, or to restore the frames as they are digitized from film and before they are converted to video. If the sequence has already been recorded, fields containing information from adjacent frames could presumably be detected and fixed. Repeated frames could be simply detected by calculating frame differences before motion compensation: frames with very small error would be assumed to be repeats and one frame would be ignored. These methods require more computation, but an added advantage is they would reduce the number of frames to process per second from 30 to 24. PAL format video does not have these problems because its display rate is 25 frames per second, and therefore contains exactly one film frame per video frame.

For the real-time implementation on the MX, a frame difference comparison before motion estimation is impractical because of the increased computation and memory bandwidth required. It was found that using an input resolution of 320 × 240 solved the interlacing problem because the system used only even fields, and the decrease in output resolution was not apparent. Repeated frames were still a problem, which appeared as unrestored damage every four frames. A less than perfect solution was to increase the computation required so processing dropped to 15 frames per second, eliminating repeated frames.

Chapter 7

Conclusions and Future Work

In this thesis, a system for removing the effects of scratches and dirt on damaged motion pictures in real-time is presented. A description of various algorithms to perform restoration is given along with performance results from using these methods to repair both simulated and real damage. The algorithms which offered the best results at a low computational cost were selected for implementation on a Precision MX video processing card. Details of the implementation and its performance when restoring actual damaged film sequences is presented.

The restoration process consists of three main steps: motion estimation and compensation, defect detection, and damage concealment. Typically, a frame is restored by using the previous and next frames in the sequence, compensating them for the effects of motion, and using the information in the surrounding frames to both detect the presence of damage and to conceal the effects of that damage. The accuracy of restoration depends on how well these steps are performed.

A detailed description of motion estimation algorithms is given in Chapter 2. It was decided that a block matching technique is best suited to real-time implementation. Various methods of searching for block matches are presented, as well

as ways to improve accuracy such as acceptance thresholds. These techniques are analyzed in detail in Chapter 5 by applying them to sequences containing simulated damage. A quantitative comparison is performed on the basis of motion estimation accuracy and computational cost.

In Chapter 3, a number of methods for detecting damage are described. The spike detection heuristics and rank ordered difference methods are shown to be the only methods with the computational efficiency to run in real-time. A further comparison of detection accuracy is performed in Chapter 5 on simulated damage. Three detection algorithms are selected as appropriate for a real-time restoration system.

Methods for concealing damage are explained in Chapter 4. Several algorithms are available which do an excellent job of concealing frame damage, but only the simpler averaging and median filters could be implemented in a real-time system. Again, a detailed comparison is performed in Chapter 5 and filters with purely temporal support (in the previous, current and next frames) are selected for implementation.

The discussion in Chapter 6 describes the implementation of the selected algorithms on the Precision MX DSP board, and especially the algorithms for the TMS320C80 processor. The graphical user interface used to control the system is described and its features explained. Finally, some performance results are obtained by using the real-time implementation to restore digitized film sequences containing real damage, as well as damaged film sequences recorded on videotape.

The restoration system is able to process a greyscale NTSC video signal at 30 frames per second at a resolution of 320 × 240 pixels. Higher resolutions and more complex restoration algorithms were also implemented, but these have a higher

computational cost. In most cases, complex restoration can be performed at over 24 frames per second, faster than the film display rate. A real-time film restoration system is therefore possible using these more advanced algorithms, but limited by the video input signal.

Most input sequences are faithfully restored by the system, and the effects of most damage are reduced. There are still problems, however, with some sequences due to inaccurate motion estimation or damage detection. These artifacts and some methods to improve them were discussed in Section 6.3.3, and more enhancements are included in the following sections.

7.1 Performance Improvements

The restoration system is able to repair film damage in real-time, but some of the more complex settings, such as a large frame size or large search window, require too much computation to finish in one frame period. One solution to this problem would be to use a faster processor. Currently, there are 60 MHz C80 processors available, compared with 40 MHz for the current implementation—a 50 percent increase in processing speed. The MX also uses slower memory (DRAM), while the C80 can support faster synchronous memory (SDRAM) [35]. It is quite probable that with faster components the system could run at least 20 percent faster, making a wider variety of algorithms suitable for processing at NTSC video rates (30 frames per second).

7.1.1 Algorithm Enhancements

A major problem for the real-time restoration system is frame rate conversion. If the source media displays fewer frames per second than NTSC video, some frames must be repeated to increase the number of displayed frames per second while keeping the overall speed of the film constant. Detecting repeated frames is possible by calculating frame differences before motion compensation, but requires too much computation and memory bandwidth for the current implementation to perform in real-time.

Knowing where a scene boundary occurs can improve restoration performance because the system can either disable restoration or use an alternate method for these frames. Algorithms are available to detect scene changes in a video signal [41, 42], but again these methods are too time consuming to perform in the present system.

Another enhancement which has not been explored is using more than one frame in the backward and forward directions. More frames could increase detection and interpolation quality, but the chances of motion estimation errors grow as the time difference between frames increases. According to [15], using more frames for support does not yield significant improvements. This is an area for future study.

7.1.2 Hardware Implementation

Now that a real-time restoration system has been successfully implemented entirely in software, a logical extension is to create a hardware implementation of similar algorithms. Such a system would undoubtedly be faster than the current implementation, allowing for higher input resolutions or more complicated restoration algorithms.

The most computationally intensive part of the restoration process is motion estimation. There are already several manufacturers producing hardware motion estimation processors for video compression applications [43, 44], and it would be a simple matter to use one of these for a restoration system. More exotic and time-consuming estimation techniques such as overlapping block search are possible if the silicon is available. Implementing other algorithms such as median or morphological filters also become much more practical when implemented in hardware.

7.2 Extensions of this Work

The current restoration system repairs only "random" types of damage such as dirt and scratches. A worthwhile future addition would be line scratch removal, as well as removal of other types of film artifacts such as jitter or brightness variance [2].

Color films could be processed to remove dirt and scratches using the same techniques as for greyscale sequences. Assuming the color film has been converted to luminance and chrominance components, the motion estimator and defect detector operate on the luminance component alone. Once damage locations have been identified, a concealment filter is applied to each channel to create the restored color frame. Other methods have been developed which treat color sequences as a special case [3].

In its current implementation, the real-time restoration system uses a video signal for input, so its maximum processing size is 640×480 pixels. The techniques developed for processing these lower resolution sequences could also be applied to full-resolution digitized film with dimensions on the order of 2000 or 4000 pixels. In fact, a number of C80 processors with the current algorithms could be used in parallel to restore larger sequences, assuming the frame information was distributed

properly to all processors. While this is possible, a custom hardware implementation would likely be cheaper and more practical.

The restoration algorithms could also be applied to other areas other than dirt and scratch removal. One possibility is the reconstruction of video streams (such as MPEG) damaged during transmission [45]. The decoded stream will contain missing regions where the macroblocks were damaged. Error coding would exactly identify the damaged locations, so detection is not necessary—motion compensation in the surrounding frames and a concealment algorithm could presumably reconstruct the missing blocks.

It is noteworthy that the algorithms used in restoration are fairly similar in structure to a block-based video compression algorithm such as MPEG. Motion estimation is performed, and motion compensated blocks are loaded into the C80 for processing (detection and concealment). This is essentially the same sequence required by the MPEG coding method—instead of performing detection and concealment, interframe coding could be done. The real-time restoration implementation therefore provides a good framework for a video compression system.

Expanding further on the similarities between video compression and restoration, the detection and concealment algorithms could be combined with a video compression system and performed immediately before block coding with a minimum of extra complexity. The quality and compression ratio of the coded video stream should be significantly better when encoding damaged film sequences.

References

[1] Michael Friend. Film/digital/film. *Journal of Film Preservation*, XXIV(50), March 1995.

[2] Y. Wu and D. Suter. Historical film processing. *Proceedings of the SPIE*, 2564:289–300, July 1996.

[3] A. C. Kokaram. *Motion Picture Restoration*. Springer-Verlag, 1998.

[4] H. Muller, W. Plaschzug, and K. Glatz. 3D interpolation for the digital restoration of 35mm film. *Proceedings of the SPIE*, 3309:287–296, January 1998.

[5] Snell & Wilcox. Online product literature. `http://www.snellwilcox.com/`.

[6] Digital Vision. Online product literature. `http://www.digitalvision.se/`.

[7] A. C. Kokaram. *Motion Picture Restoration*. PhD thesis, Cambridge University, May 1993.

[8] Peter Eaves. Traditional film restoration techniques in a digital world. *SMPTE Journal*, pages 224–225, April 1998.

[9] R. D. Morris. *Image Sequence Restoration Using Gibbs Distributions*. PhD thesis, Cambridge University, May 1995.

[10] A. Kokaram, P. Rayner, P. van Roosmalen, and J. Biemond. Line registration of jittered video. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 2553–2556, April 1997.

[11] Richard Storey. Electronic detection and concealment of film dirt. *SMPTE Journal*, pages 642–647, June 1985.

[12] B. Fisher. Imaging to the rescue: The digital restoration of Disney's Snow White. *Advanced Imaging*, 8(9), September 1993.

[13] O. Buisson, B. Besserer, and S. Boukir. Deterioration detection for digital film restoration. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 78–84, June 1997.

[14] D. Suter and P. S. Richardson. Historical film restoration and video coding. In *Proceedings of PCS 96*, pages 389–394, March 1996.

[15] A. C. Kokaram, R. D. Morris, W. J. Fitzgerald, and P. J. W. Rayner. Interpolation of missing data in image sequences. *IEEE Transactions on Image Processing*, 4(11):1509–1519, November 1995.

[16] M. N. Chong, W. B. Goh, S. Kalra, and D. Krishnan. A multiprocessor motion picture restoration system. *International Journal of Information Technology*, 3(1):1–16, 1997.

[17] Gerhard Wischermann. The digital wetgate: A third-generation noise reducer. *SMPTE Journal*, pages 95–100, February 1996.

[18] Paul Read. New technologies for archive film images restoration and access. *Image Technology*, 78(8), September 1996.

[19] A. N. Netravali and B. G. Haskell. *Digital Pictures: Representation, Compression, and Standards.* Plenum Press, second edition, 1995.

[20] A. C. Kokaram, R. D. Morris, W. J. Fitzgerald, and P. J. W. Rayner. Detection of missing data in image sequences. *IEEE Transactions on Image Processing*, 4(11):1496–1508, November 1995.

[21] Frédéric Dufaux and Fabrice Moscheni. Motion estimation techniques for digital TV: A review and a new contribution. *Proceedings of the IEEE*, 83(5):858–876, June 1995.

[22] J. Magarey, A. Kokaram, and N. Kingsbury. Robust motion estimation using chrominance information in colour image sequences. *Image Analysis and Processing*, 2:486–493, September 1997.

[23] R. W. Young and N. G. Kingsbury. Frequency-domain motion estimation using a complex lapped transform. *IEEE Transactions on Image Processing*, 2(1):2–17, January 1993.

[24] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing.* Prentice Hall, 1989.

[25] Vasudev Bhaskaran and Konstantinos Konstantinides. *Image and Video Compression Standards: Algorithms and Architectures.* Kluwer Academic Publishers, 1995.

[26] Bede Liu and André Zaccarin. New fast algorithms for the estimation of block motion vectors. *IEEE Transactions on Circuits and Systems for Video Technology*, 3(2):148–157, April 1993.

[27] M. Bierling. Displacement estimation by hierarchical blockmatching. *SPIE Visual Communications and Image Processing*, 1001:942–951, 1988.

[28] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley, 1993.

[29] A. Kokaram and P. Rayner. A system for the removal of impulsive noise in image sequences. In *Proceedings of SPIE*, pages 322–331, November 1992.

[30] M. J. Nadenau and S. K. Mitra. Blotch and scratch detection in image sequences based on rank ordered differences. In *5th International Workshop on Time-Varying Image Processing and Moving Object Recognition*, pages 10–15, Florence, Italy, September 1996.

[31] J. P. Tremblay and P. G. Sorenson. *An Introduction to Data Structures with Applications*. McGraw-Hill, second edition, 1984.

[32] Simon Haykin. *Adaptive Filter Theory*. Prentice Hall, third edition, 1996.

[33] S. Geman, D. E. McClure, and D. Geman. A nonlinear filter for film restoration and other problems in image processing. *CVGIP: Graphical Models and Image Processing*, 54(4):281–289, July 1992.

[34] Precision Digital Images Corporation. *Precision MX Video Engine: Technical Reference*, April 1996.

[35] Texas Instruments Incorporated. *TMS320C80 Digital Signal Processor Data Sheet*, March 1996.

[36] Texas Instruments Incorporated. *TMS320C80 (MVP) Master Processor User's Guide*, 1995.

[37] Texas Instruments Incorporated. *TMS320C80 (MVP) Parallel Processor User's Guide*, 1996.

[38] Texas Instruments Incorporated. *TMS320C80 (MVP) Transfer Controller User's Guide*, 1995.

[39] Texas Instruments Incorporated. *TMS320C80 (MVP) Video Controller User's Guide*, 1995.

[40] Texas Instruments Incorporated. *TMS320C80 (MVP) Multitasking Executive User's Guide*, 1995.

[41] C. Taskiran and E. J. Delp. Video scene change detection using the generalized sequence trace. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1998.

[42] Y. Wu and D. Suter. A comparison of methods for scene change detection in noisy image sequence. In *Proceedings of VISUAL 96*, pages 459–467, Melbourne, Australia, 1996.

[43] C. Chiu, W. Hui, T. J. Ding, and J. V. McCanny. A 64-point Fourier transform chip for video motion compensation using phase correlation. *IEEE Journal of Solid-State Circuits*, 31(11):1751–1761, November 1996.

[44] P. Pirsch, N. Demassieux, and W. Gehrke. VLSI architectures for video compression—a survey. *Proceedings of the IEEE*, 83(2):220–246, February 1995.

[45] P. Salama, N. Shroff, and E. J. Delp. A Bayesian approach to error concealment in encoded video streams. In *Proceedings of the International Conference on Image Processing*, pages 49–52, Lausanne, Switzerland, September 1996.

[46] Mark-Paul Meyer. Ethics of film restoration using new technology. *Image Technology*, 78(8), September 1996.

[47] Library of Congress, Washington D.C. The national film preservaton plan: An implementation strategy. *Journal of Film Preservation*, XXIV(51), November 1995.