Motion Modeling and Segmentation in Compressed Video with Applications

by

Yue Meng Chen

M.A.Sc., Simon Fraser University, 2007
B.Sc., Zhejiang University, 1998

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy

in the
School of Engineering Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY

Spring 2012

Approval

Name: **Yue Meng Chen**

Degree: **Doctor of Philosophy**

Title of Thesis: ***Motion Modeling and Segmentation in Compressed Video with Applications***

Examining Committee:

Chair: **Dr. Shawn Stapleton, P. Eng**
Professor
School of Engineering Science

Dr. Ivan V. Bajić, P. Eng
Senior Supervisor
Associate Professor, School of Engineering Science

Dr. Mirza Faisal Beg, P. Eng
Supervisor
Associate Professor, School of Engineering Science

Dr. Parvaneh Saeedi, P.Eng
Supervisor
Assistant Professor, School of Engineering Science

Dr. Greg Mori
Associate Professor
School of Computing Science

Dr. Ling Guan
Professor
Department of Electrical and Computer Engineering
Ryerson University

Date Defended/Approved: April 16, 2012

Partial Copyright Licence

Abstract

This Ph.D. thesis offers a perspective of the theory, applications, and implementation of motion modeling and segmentation in compressed video. The research effort is devoted to retrieving object motion information from compressed video and producing images with desired spatial and temporal resolutions.

On the theoretical side, this research project attempts to model motion and retrieve substantial information about objects in 3D real-life scenes from compressed 2D images, and develop techniques to reconstruct an image that fits into the same perspective of the sequence with desired temporal and spatial resolutions, which might or might not conform to those of the original sequence.

From an application perspective, three types of scenarios are then classified: 1) frame prediction, 2) frame interpolation and 3) frame resizing. Scenario 1) involves temporal causal processing where video data from the past is used, while scenario 2) is temporal, noncausal processing where both past and future video data are exploited. A variety of video applications can be found in either of these two scenarios, e.g., predictive decoding for delay reduction, whole-frame error concealment, playout buffer control for video streaming, frame rate up-conversion, and so on. Scenario 3) is spatial processing for producing a frame with desired spatial resolution, e.g., super-resolution reconstruction of compressed video.

State-of-the-art block-based video codecs are targeted for the implementation and evaluation of the effectiveness of the proposed system.

Keywords: Motion Modelling; Object Segmentation; Frame Synthesis; Super-Resolution; Predictive Decoding of Video; Frame Rate Up-Conversion;

To my parents Zhifu Chen and Baodi Zhou, and
my beloved wife Yu Gu.

Acknowledgements

My PhD years at Simon Fraser University and this thesis have had mentorship from many outstanding individuals from academia and industry. I have my heartfelt gratitude and thanks to them, for without their help, this thesis would not have seen its completion date.

First and foremost I would like to thank my passionate and dedicated committee members, *Ivan V. Bajić*, *Pavaneh Saeedi*, *Mirza Faisal Beg*, *Shawn Stapleton* (School of Engineering, SFU), *Greg Mori* (School of Computer Science, SFU), and *Ling Guan* (Department of Electrical and Computer Engineering, Ryerson University), without whose support this research could not have been done.

Over the years, *Prof. Ivan V. Bajić*, my senior supervisor, has taught me so much about digital signal processing, image and video processing and compression, multimedia communications, guided me on identifying research fields, and offering in-deep advice throughout my Master and PhD years. I also thank him for giving me the freedom of exploration in research. I like to in particular thank *Prof. Pavaneh Saeedi*, for her help on solving motion segmentation problems, and numerous other digital image processing problems. I am grateful most of all for her confidence in me as well as her steady guidance. The analysis phase of the research called for the fundamental understanding on machine learning and computer vision. I would like to show my gratitude to *Prof. Greg Mori*, for his expertise and help in this field. I also like to thank *Prof. Jie Liang*, for all his help on video compression and digital signal processing. He guided my study towards master degree, and discussed and shaped my ideas for the PhD research. I would also thank *Prof. Mirza Faisal Beg* and *Prof. Ling Guan* for the patience to read this thesis and contributing to its content. I like to thank *Prof. Shawn Stapleton* for chairing the defence.

I would especially like to thank my long time friend and mentor *Dr. Chi-Shin Wang*, for his tremendous help during my PhD years.

I feel very privileged to have been allowed to get to know and learn from all my friends and fellow graduate students in their field during my research studies. I am especially

grateful to *Jing Wang*, *Choong-Hoon Kwak*, *Upul Samarawickrama*, *Hadi Hadizadeh*, *Carl Qian*, *Xiaonan Ma*, *Sohail Bahmani*, *S. Mohsen Amiri*, *Duncan Chan*, *Guoqian Sun*, *Calvin Che*, for helping me get through the difficult times, and for all the emotional support, entertainment, and caring they provided..

Lastly, and most importantly, I. would like to thank my parents and my wife *Victoria*, who through my study career had always encouraged me to follow my heart and chase my dreams. I am forever indebted to *Victoria* for her understanding and endless patience when it was most required, and all the love and support she has given me during the thesis effort.

Table of Contents

List of Tables

List of Figures

List of Symbols

$\cdot\mathbf{a}_{i_i}$	Parameter array in the least squares predictor
$\cdot\beta$	Homogeneity controlling parameter for moving region segmentation
$\cdot\lambda_{MOTION}$	Lagrange parameter for mode selection in H.264
$\cdot\Delta_k$	k-th directional gradient component
$\cdot\Delta\mathbf{m}$	The difference between estimated and true $\mathbf{m}$
$\cdot BD_R$	Boundary discontinuity value of region R
$\cdot\mathbf{c}$	Coordinates of the block center
$\cdot D_{mag}$	Average relative difference in magnitude
$\cdot D_{ph}$	Average relative difference in phase
$\cdot D^R$	Motion vector deviation in region R
$\cdot\mathbf{e}$	Frame estimation error vector
$\cdot E[\cdot]$	Expectation operator
$\cdot E_{\mathbf{m}}$	Squared L_2 norm of the difference between estimated and true $\mathbf{m}$
$\cdot E_{res}$	Sum of squared prediction residuals
$\cdot\mathbf{f}^{curr}$	Current frame
$\cdot\mathbf{f}^{prev}$	Previous frame
$\cdot\hat{\mathbf{f}}$	Estimated frame
$\cdot h_i$	Distance between two pixels
$\cdot\mathbf{H}(\mathbf{m})$	Homographic mapping matrix
$\cdot I_j^R$	Region motion deviation on the j-th pixel in region R
$\cdot J$	The Lagrangian
$\cdot\kappa$	Constant factor
$\cdot\mathbf{m}$	Vector of global motion model parameters
$\cdot\mathbf{MV}_{x,y}(t)$	Motion vector of a block located at (x,y) at time t
$\cdot\mathbf{MV}_{x,y}(t;\mathbf{m})$	Motion vector of a block located at (x,y) at time t synthesized using global motion model $\mathbf{m}$
$\cdot MV^{res}$	Residual MV field after global motion compensation
$\cdot\mathbf{MV}_i$	The i-th motion vector
$\cdot\mathbf{MV}_{cent}^R$	Centroid MV of region R
$\cdot OV_k(x,y)$	The pixel at location (x,y) in the k-th overlapping block

$\cdot P(\omega)$	Probability of label ω in a Markov random field
$\cdot P(x, y)$	Pixel value at coordinates (x, y)
$r(x, y)$	Prediction residual at coordinates (x, y)
$\cdot R_i^{t \to t+m}$	The translation of the i-th moving region from the image plane at time t to $t + m$
$\cdot R_i$	The i-th moving region
$\cdot S^R$	The scale factor for a pixel located in region R
$\cdot S_i^j$	The 3×3 neighbourhood of inlier MVs
$\cdot T$	Temporal/Spatial operator for frame synthesis
$\cdot T_{mag}^i$	Maximum relative magnitude difference
$\cdot T_{ph}^i$	Maximum relative phase difference
$\cdot TH$	A threshold
$\cdot \omega_i$	The i-th Markov Random Field label
$\cdot \varphi_x$	Rotational angle on x-axis
$\cdot \varphi_y$	Rotational angle on y-axis
$\cdot \varphi_z$	Rotational angle on z-axis
$\cdot V(C)$	Potential of clique C
$\cdot v_{hori}$	Horizontal velocity
$\cdot v_{vert}$	Vertical velocity
$\cdot (x, y)$	Pixel coordinate in the current frame
$\cdot (x', y')$	Pixel coordinate in the reference frame
$\cdot \bar{y}, \bar{u}, \bar{v}$	Mean value of Y/U/V
$\cdot w_y$	Weights applied to Y component (or U/V component)
$\cdot \boldsymbol{\mu}_{\mathbf{MV}}$	Mean vector of a set of MVs
$\cdot \sigma_{\mathbf{MV}}^2$	Standard deviation of MV magnitudes from the mean vector
$\cdot \oplus$	Aggregation of pixels from multiple moving regions

List of Acronyms

BMA	Block Matching Algorithm
CIF	Common Interchange Format
DCT	Discrete Cosine Transform
ES	Exhaustive Search
FAR	False Alarm Rate
FMV	Forward Motion Vector
FN	False Negative
FP	False Positive
FRUC	Frame Rate Up Conversion
GD	Gradient Descent
GM	Global Motion
GME	Global Motion Estimation
GMC	Global Motion Compensation
ICM	Iterated Conditional Modes
ISO	International Standards Organization
ITU	International Telecommunications Union
LM	Local Motion
LMMSE	Linear Minimum Mean Square Error
LR	Low Resolution
LSS	Least Squares Solution
LSS-ME	Least Squares Solution M-Estimator
MAD	Mean Absolute Difference
MB	Macroblock
MC	Motion Compensation
MCFI	Motion-Compensated Frame Interpolation
ME	Motion Estimation
MPEG	Moving Picture Experts Group
MRF	Markov Random Field
MSE	Mean Square Error
MV	Motion Vector
MVF	Motion Vector Field

OBMC	Overlapped Block Motion Compensated
PSNR	Peak Signal-to-Noise Ratio
QCIF	Quarter Common Interchange Format
RANSAC	RANdom SAmple Consensus
RDO	Rate Distortion Optimized
RBFI	Region-Based Frame Interpolation
RBPD	Region-Based Predictive Decoding
RMD	Region Motion Deviation
RMSE	Root-Mean-Square Error
SIF	Standard Interchange Format
SAD	Sum of Absolute Differences
SSD	Sum of Squared Differences
TN	True Negative
TP	True Positive
SR	Super-Resolution

1. Introduction

1.1. Background and Motivation

This Ph.D. work focuses primarily on **motion modelling and segmentation in compressed video with applications**, that is, retrieving local and global motion information for moving objects from compressed video, and applying it to synthesize video frames in a desired temporal and spatial resolution.

Video nowadays serves much more than a way of entertaining. Its applications have reached out to many fields, such as security and communications, and demands for good quality and manageability have been increasing accordingly. Video compression is often used to facilitate both storage and transmission. To manage the explosive growth of video compression techniques, the ISO and ITU-T have been actively standardizing video compression since 1980s, and have come up with various standards, such as MPEG-1, MPEG-2, MPEG-4, H.261, H.263 and H.264.

State-of-the-art video compression approaches are mostly constructed with a block-based architecture, i.e. the input image is partitioned into small blocks, and temporal/spatial redundancies are removed on a block basis for compression purposes. In the MPEG-4 standard, an attempt is made to compress video using a content-based approach. However, object-based video compression has not been successful yet due to the difficulty of video segmentation.

In the compressed video bitstream produced by conventional encoders, all temporal/spatial information, such as motion vectors (MVs) and transform coefficients, are associated with particular blocks rather than objects. The motivation behind this Ph.D. work is to understand how compressed data is related to object characteristics such as motion, shape, texture and colour, in real-life video, and to contribute to the

algorithmic development and architectural implementation of transmission, storage, and rendering of compressed video in an object-cognizant way.

This Ph.D. thesis explains various proposed methods for modeling object and camera motion in compressed video, segmenting moving objects, and synthesizing video frames in a desired temporal and spatial resolution. In the next section, a brief preview of the previous work on motion modelling, object segmentation and frame synthesis is provided .

1.2. Related Work

1.2.1. Motion Modelling in Compressed Video

Motion modelling is an important aspect in modern video coding standards, which are all structured on the basis of blocks. The widely used translatory motion model assigns one motion vector (MV) to a given block. This MV is obtained through motion estimation (ME), and utilized by motion compensation (MC) to remove temporal redundancy. Motion modelling and estimation techniques have undergone significant improvement throughout multiple generations of video coding standards. Balancing their coding efficiency and estimation accuracy has stood at the centre of video coding research in the last few decades.

Among all motion estimation techniques, block matching algorithms (BMA) are the main approach to obtain MVs for the purpose of compression. While the basic idea behind BMA has remained more or less the same, the block partitioning has evolved from the fixed block size of 16×16 pixels (H.261) to adaptive block sizes as low as 4×4 pixels in the latest standard (H.264). The BMA leads to a two-dimensional displacement vector, commonly called a motion vector (MV). As the main goal of motion estimation is to improve compression efficiency, the final MVs do not always represent true motion. This inaccuracy becomes one of the sources of visual artefacts on the decoder side when a frame gets reconstructed, and becomes more pronounced when frame prediction is performed, for example in predictive decoding, as will be illustrated later in the text. We provide a detailed literature review on motion modeling and estimation in Chapters 2 and 4.

1.2.2. Video Object Segmentation

Video object segmentation is a useful technique for object-based media representation and description (e.g. MPEG-4 and -7). In recent years, object segmentation and tracking have gained more attention due to emerging content-based multimedia applications, such as video indexing and retrieval, video surveillance, and interactive video applications. Even before MPEG-4 introduced the concept of treating scenes as compositions of audio-visual objects, there have been intensive research efforts on image and video object segmentation. Especially interesting is the problem of moving region segmentation in compressed video, due to the abundance of compressed video content.

Depending on the segmentation cues exploited in processing, state-of-the-art segmentation approaches related to compressed video can be broadly classified into three types: 1) segmentation in the pixel domain, which mainly uses reconstructed image features (e.g. colour, edge and texture), 2) segmentation in the "compressed domain," which only utilizes compressed video data (e.g. MVs and transform coefficients), and 3) joint (or hybrid) segmentation methods, which combine both pixel-domain and compressed-domain information to balance speed and accuracy. In a video recorded by a moving camera, the estimated MVs represent a composition of true object motion and global (or camera) motion. In such cases, it is important to estimate and compensate global motion prior to object segmentation. Global motion estimation can be conducted in either the pixel domain or the compressed domain.

Since video data is almost always stored in compressed form, compressed domain segmentation appears more attractive in some real-time applications, since it can considerably reduce processing time and memory requirements. However, working with compressed data usually implies reduced segmentation accuracy, and it remains a challenge to devise a fully compressed domain segmentation method that achieves accurate object boundaries. In Chapters 3 and 4, we present a detailed literature review on object segmentation.

1.2.3. Frame Prediction

Frame prediction is the technique of predicting a future frame based on spatial and/or temporal information. A particular application of interest in this work is predictive video decoding. Such a scheme allows the decoder to predict and display a frame that has not yet arrived at the receiver. This technique can be useful in many video applications, such as delay reduction in interactive video communications, jitter buffer management in video streaming, and the concealment of whole-frame losses. Conventional frame prediction approaches, such as motion extrapolation, heavily rely on motion information extracted from the compressed video stream. However, since motion vectors are not always accurate enough to represent true motion, annoying artefacts may appear in predicted frames, such as background shaking and object boundary distortion.

A related application studied in the past is whole-frame concealment. The methods for whole-frame concealment can be broadly divided into two categories: 1) pixel-based, and 2) block-based. One drawback of pixel-based whole-frame concealment is its computational complexity. Compared to pixel-based approaches, block-based approaches have a significantly lower computational complexity, because they process less information, typically only one MV per block of pixels. However, conventional MV prediction by median filtering of MVs from a neighbourhood may disrupt the spatio-temporal relationships among the MVs of neighbouring blocks, and thus lead to annoying visual artefacts. Moreover, in order to produce better MVs, a motion re-estimation on the decoder side has also been studied in the past; however, accuracy is greatly affected by the amount of extra computation introduced at the decoder. Related literature review is further given in Chapters 5 and 6.

1.2.4. Temporal Frame Interpolation

Frame interpolation generally describes the techniques of interpolating a frame based on spatial and/or temporal information. One well-known application is frame rate up-conversion (FRUC), whose goal is to increase temporal resolution by interpolating frames between existing video frames. It has been proposed as a tool for video communications, where video sources sometimes have to be temporally downsampled to meet low bandwidth requirements of communication channels. FRUC also plays a

role in other video applications, such as display format conversion and slow motion playback. Various FRUC algorithms have been developed in the past. Non-motion compensated schemes, such as frame repetition and frame averaging, produce a frame by repeating or combining the pixel values at the same location between neighbouring frames. These schemes are fast, but lead to blurring and/or motion jerkiness when strong motion is present in the scene.

To improve visual quality, a frame can be interpolated based upon the motion vectors (MVs) between two consecutive frames, leading to motion-compensated frame interpolation (MCFI). Due to the low density of block-based MVs and their inaccuracies, direct MCFI causes annoying visual artefacts in interpolated frames. Many approaches have been proposed to mitigate these artefacts, including vector median filtering, overlapped block motion compensated (OBMC) interpolation, multistage refinement, and bilateral and global motion estimation. Alternatively, instead of using block-based motion, region-based schemes interpolate frames by considering moving regions or objects. The basic idea is to characterize the region motion using an affine or perspective motion model and use such a region-based motion model to synthesize an intermediate frame. Related literature review is given in Chapters 5 and 6.

1.2.5. Spatial Frame Interpolation

Spatial frame interpolation or frame resizing describes the techniques for increasing or decreasing frame resolution. One challenging research topic for frame resizing is super-resolution (SR), which is a technique for enhancing the resolution of an imaging system. There are both single-frame and multiple-frame variants of SR, where multiple-frame approaches are most powerful. Algorithms can also be divided by their domain: frequency or space domain. By fusing together several low-resolution (LR) images one enhanced-resolution image is formed. The SR approaches proposed in the literature are reviewed in detail in Chapters 5 and 6.

1.3. Scope of the Thesis

The goal of this research work is to develop methods for motion modeling and segmentation from compressed video and their use in various applications. Objects in a video scene can be broadly classified as background and foreground objects. Each object can further consist of several regions with homogeneous features, such as motion and colour. Therefore, each video frame can be seen as a composition of video objects. In this work, these objects are obtained from the compressed video stream and offer a possibility to composite (synthesize) a non-existing, larger, or future frame under various circumstances.

The concept of content-based frame composition is illustrated in Fig. 1.1. The received video bitstream is fed to a syntactic decoder that parses information based on syntactic structures of the particular video coding standard. The elementary information such as MVs and transform coefficients, are extracted from the bitstream by this syntactic decoder and passed to the standard video decoder to reconstruct frames. At the same time, the compressed domain information is used to model sequence features, such as local object motion and camera motion. Combining all the information, a frame is finally assembled by a compositor at the desired temporal and spatial scale for video display.

The frame composition model (prediction and interpolation) is constructed under two main assumptions: 1) regions move in a coherent fashion over a short period of time, and 2) when two regions move towards the same location, the region closer to the view point will occlude the region further away from the view point. The background region is considered to have the largest distance to the view point.

The central part of this thesis is object motion modelling and segmentation in compressed video. Also described are various frame synthesis approaches under different circumstances, which lead to a range of applications, such as temporal frame prediction and interpolation and spatial frame resizing. The overall goal is to find low complexity solutions, capitalizing on the information already present in the compressed video stream, yet achieve superior visual quality of the synthesized video frames.

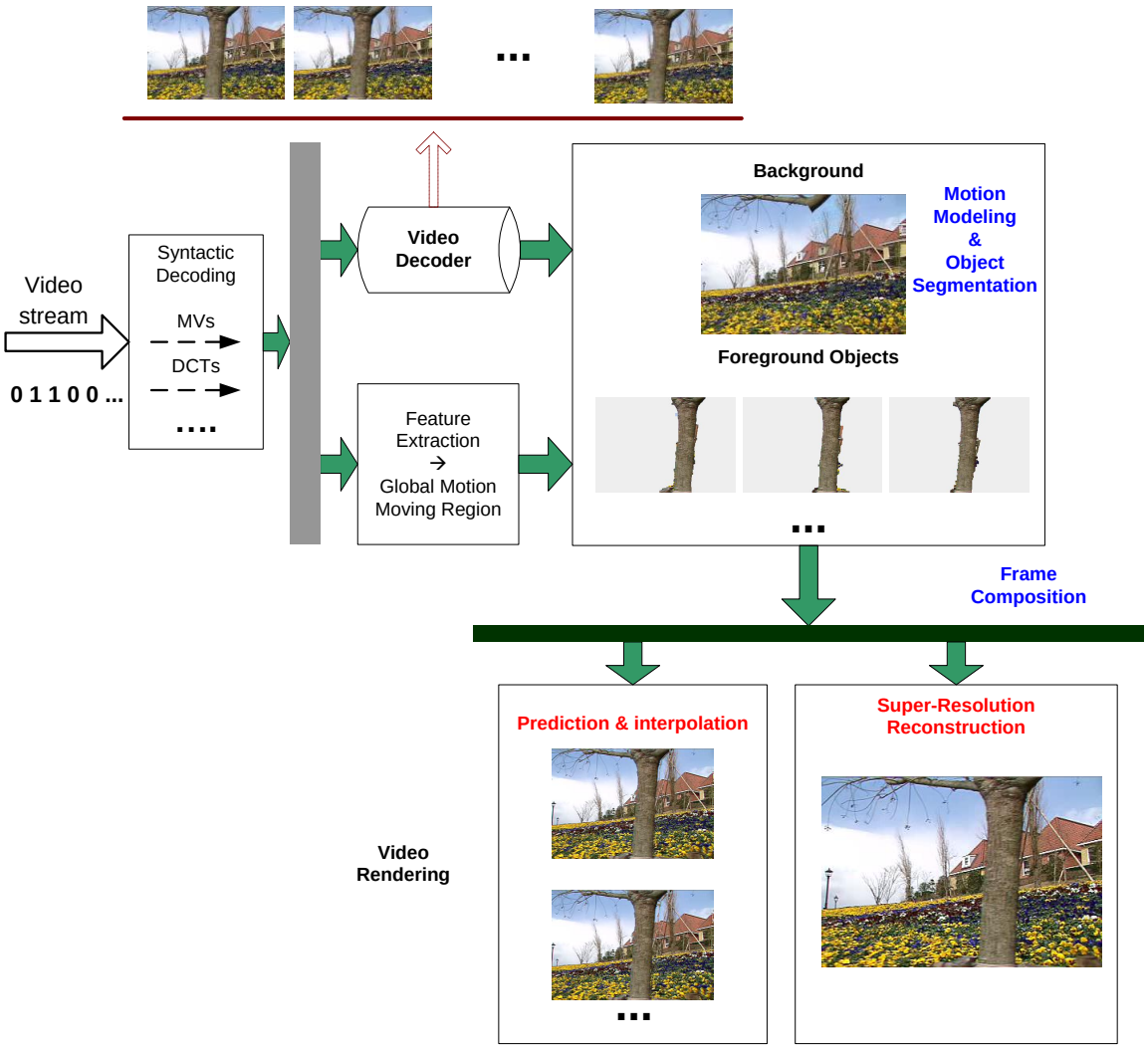


Figure 1.1: Motion modeling and segmentation with their applications in frame prediction, interpolation, and super-resolution.

1.4. Contributions

In this work, we developed multiple approaches for global motion estimation and object motion segmentation, as well as a novel framework for joint global motion estimation and object segmentation. While many related methods have been studied before in the pixel domain, the main novelty of the present work lies in adapting these methods to compressed-domain video processing, and addressing certain issues that arise in dealing with compressed video data. Substantial emphasis is placed on reusing the data available in a compressed video bitstream, such as motion vectors and prediction residuals, whenever possible. Other contributions of the thesis include adapting the developed region-based compressed-domain methods to various applications, such as frame prediction, interpolation, and resizing.

Specifically, the contributions to the knowledge and practice in the field of video processing include the four methods listed below. In each case, the specific novelty of the method and its comparison with state-of-the-art is presented in detail in the corresponding chapter.

- A cascade-of-rejectors approach for removing block-based MV outliers in global motion estimation (Sections 2.2.3—2.2.5).
- An unsupervised segmentation algorithm for extracting moving regions from compressed video using a Markov Random Field (MRF) model for block-based MVs (Section 3.4).
- A joint framework of simultaneous moving object segmentation and global motion estimation from block-based MVs (Chapter 4).
- Region-based predictive decoding for end-to-end delay reduction in video communications (Sections 6.1.2—6.1.4).

Overall, the research conducted in the past 5 years has resulted in the following 12 academic publications, comprising 4 journal papers and 8 conference papers:

Journals:

[1]. **Y.-M. Chen** and I. V. Bajić, "A joint approach to global motion estimation and motion segmentation from a coarsely sampled motion vector field," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 21, no. 9, pp. 1316-1328, Sep. 2011.

[2]. **Y.-M. Chen**, I. V. Bajić, and P. Saeedi, "Moving region segmentation from compressed video using global motion estimation and Markov random fields," *IEEE Trans. Multimedia*, vol. 13, no. 3, pp. 421-431, Jun. 2011. (Special Issue on ICME 2010)

[3]. **Y.-M. Chen** and I. V. Bajić, "Region-based predictive decoding of video," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 3, pp. 452-457, Mar. 2010.

[4]. **Y.-M. Chen** and I. V. Bajić, "Motion vector outlier rejection cascade for global motion estimation," *IEEE Signal Processing Letters*, vol. 17, no. 2, pp. 197-200, Feb. 2010.

Conferences:

[1]. **Y.-M. Chen** and I. V. Bajić, "Predictive video decoding using GME and motion reliability," *Proc. SPIE Applications of Digital Image Processing XXXIV*, Vol. 8135, San Diego, CA, Aug. 2011.

[2]. **Y.-M. Chen** and I. V. Bajić, "Spatio-temporal super-resolution from compressed video employing global and local motion," *Proc. IEEE PacRim'11*, pp. 907-912, Victoria, BC, Aug. 2011.

[3]. **Y.-M. Chen**, I. V. Bajić, and P. Saeedi, "Motion segmentation in compressed video using Markov random fields," *Proc. IEEE ICME'10*, pp. 760-765, Singapore, Jul. 2010.

[4]. **Y.-M. Chen** and I. V. Bajić, "Predictive video decoding based on ordinal depth of moving regions," *Proc. IEEE ICC'10*, Cape Town, South Africa, May 2010.

[5]. **Y.-M. Chen** and I. V. Bajić, "Compressed-domain moving region segmentation with pixel precision using motion integration," *Proc. IEEE PacRim'09*, pp. 442-447, Victoria, BC, Aug. 2009.

[6]. **Y.-M. Chen**, I. V. Bajić, and C. Qian, "Frame rate up-conversion of compressed video using region segmentation and depth ordering," *Proc. IEEE PacRim'09*, pp. 431-436, Victoria, BC, Aug. 2009.

[7]. **Y.-M. Chen**, I. V. Bajić, and P. Saeedi, "Coarse-to-fine moving region segmentation in compressed video," *Proc. IEEE WIAMIS'09*, pp. 45-48, London, UK, May 2009.

[8]. **Y.-M. Chen** and I. V. Bajić, "Predictive decoding for delay reduction in video communications," *Proc. IEEE Globecom'07*, pp. 2053-2057, Washington, DC, Nov. 2007.

1.5. Organization of the Thesis

The thesis is divided into three parts: “motion modeling,” “object segmentation,” and “applications.” The first chapter contains the introduction and some background on motion modeling, object segmentation, and their applications in frame prediction, interpolation and resizing. More detailed discussion of these topics, along with the literature review, appears in Chapters 2 through 6. A conclusion is given in Chapter 7.

In Chapter 2, we present the fundamentals of the thesis on various motion modeling approaches, and discuss the motion reliability in compressed video. Based upon motion information and reconstructed frames, in Chapter 3 we present various compressed-domain object segmentation approaches. Motion segmentation and global motion estimation are further studied in Chapter 4. As two operations greatly affect one another, a joint framework is presented to achieve simultaneous object segmentation and global motion estimation. In Chapter 5, we discuss how to compose a frame using motion and object information. The techniques presented in this chapter are later adopted in various content-based video applications to render a frame for final display. In Chapter 6, we address the applications that utilize motion estimation and segmentation techniques, broadly divided into three scenarios: frame interpolation, frame prediction, and frame resizing.

Finally in Chapter 7, we conclude this thesis, and offer future research direction in related field.

2. Motion Modelling in Compressed Video

In this chapter, we discuss the problem of motion modelling in compressed video. In Section 2.1, we describe motion structure in block-based video codecs, particularly motion estimation techniques and MV reliability. In Section 2.2, we describe how to estimate global motion from a block-based MV field. An important component of global motion estimation (GME) is outlier rejection, for which we propose a novel outlier rejection filter, and its performance is evaluated within GME framework.

2.1. Motion Structure in Block Based Video Codecs

All recent video coding standards utilize blocks of pixels for motion estimation, compensation, transform, and the control of coding parameters. These blocks are commonly referred to as macroblocks (MBs) and consist of 16×16 pixels. Motion compensation (MC) and motion estimation (ME) are two major techniques to reduce temporal redundancy in a video coding system. MC synthesizes an image block in the current frame by fetching a block of pixels from a reference frame using an estimated MV, while ME estimates the MV for a given block. Looking back over the video coding evolution route, we have witnessed video compression achievements by incorporating new motion techniques and optimizing already built-in techniques, including the first ITU (former CCITT) coding standard H.120 (1984-1988) and its block-based successors H.261 (1991), MPEG-1 (1993) [120], MPEG-2/H.262 (1994), H.263 (1996), MPEG-4 (1999) and the latest H.264 (2003) [76] [120 – 123].

2.1.1. Video Coding Standards

In MPEG video coding standards, there are three types of frames: 1) intra coded (I-) frames, 2) temporally predicted (P-) frames, and 3) bi-directionally (B-) temporally predicted frames [35]. Among these three frame types, I-frames use most bits. An I-frame is encoded as a still image, without referencing any other frames in the video.

Only spatial redundancy is reduced, largely through the discrete cosine transform (DCT), similar to JPEG coding. P- and B-frame coding involves temporal prediction. For both of these frame types, motion compensation is performed to reduce temporal redundancy. The bitstream of both P- and B-frames includes motion vectors (MVs) and DCT coefficients of the motion compensated prediction residuals. Because of temporal redundancy removal, P- and B-frames have significantly fewer bits than I- frames. B-frames usually achieve the highest compression ratio among the three frame types due to bi-directional temporal prediction.

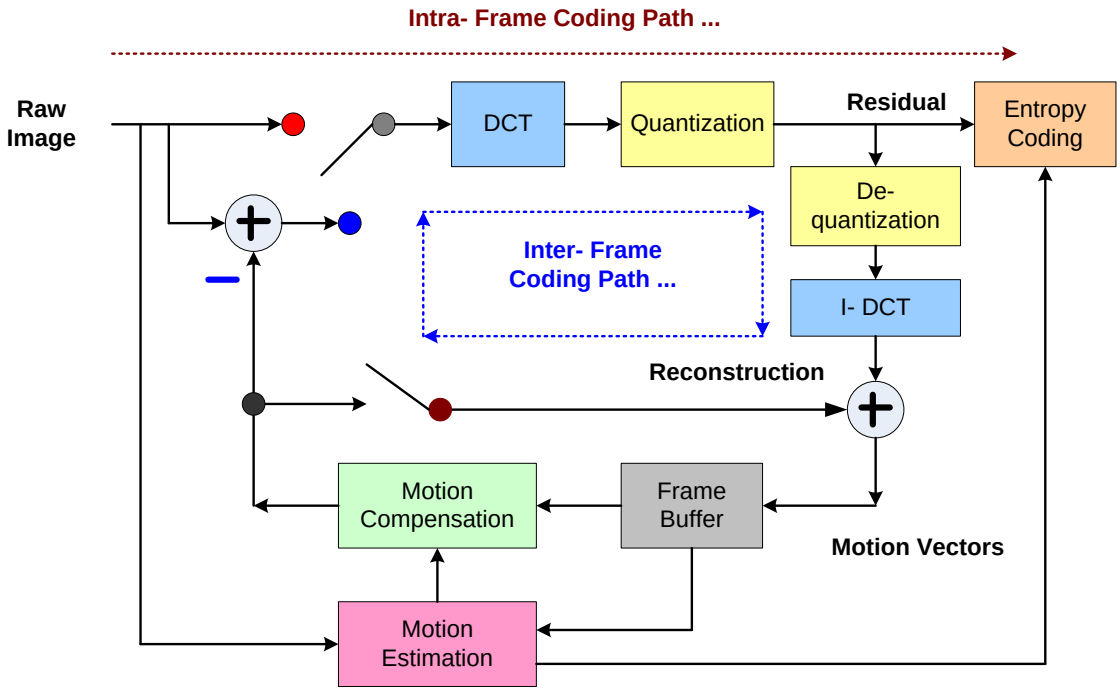


Figure 2.1: Block-based video encoder.

Figures 2.1 and 2.2 together illustrate the basic functional modules in a block-based video codec. The video encoder takes in a sequence of video frames. All the MB's in I-frames (and possibly some MB's in P- and B-frames) are intra-coded, which means they are coded without temporal prediction. However, in H.264, the latest video coding standard, intra-coded MB's are spatially predicted [36]. Inter-coded MB's, on the other hand, are coded using temporal prediction from neighbouring frames. Whichever

form of prediction is used, spatial or temporal, only the prediction residual of the corresponding block is encoded. These prediction residuals are fed into the transformer, usually based on DCT or a related transform, and then further quantized by a matrix of quantizers. Since the energy distribution after the transform is concentrated in the low frequency areas, a special zigzag mapping matrix is designed to scan the coefficients so that long runs of zeros are grouped together. The zigzag-scanned coefficients, as well as the MVs, are entropy coded.

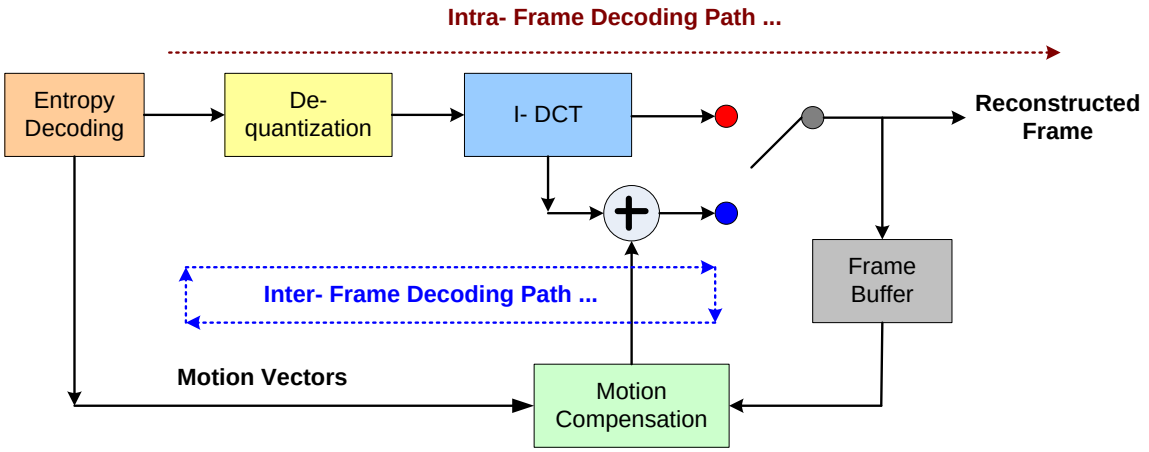


Figure 2.2: Block-based video decoder.

As the motion estimation techniques evolved over time, the block sizes in motion estimation and compensation modules have been made adaptive in order to increase the estimation accuracy. Optimal selection of these parameters is determined by the encoder in terms of coding efficiency and reconstructed video quality. The encoder determines the cost of each MB coding mode in terms of bits, which includes the cost of coding associated MVs and prediction residuals. Also, each coding mode produces a certain reconstructed signal quality. Combining these two pieces of information, the optimal coding mode is selected. The reconstructed MB refreshes the reference frame buffer to reconstruct exactly the same MB on the decoder side, while the coded bits are output into the bitstream.

The decoder structure is shown in Fig. 2.2. It consists of processing modules that perform reverse operations from the encoder except for the motion estimation and mode selection modules, which are not needed at the decoder.

2.1.2. Motion Estimation Techniques

The evolution of ME can be traced back through two paths: 1) improving the accuracy of the MVs, and 2) decreasing and varying the size of the compensated block. More specifically, translational MVs have evolved from integer-pixel in MPEG-1/H.261 to half-pixel in MPEG-2/H.263, and finally quarter-pixel in H.264; block sizes have been refined from the single 16×16 pattern in H.261, H.263, MPEG-1, and MPEG-2 to include 8×8 blocks in MPEG-4, and eventually 4×4 block and even the non-square 16×8 and 8×16 blocks in H.264. Although in theory the MV accuracy can be quantized even finer, B. Girod [37] in 1993 conducted analysis on fractional-pixel accuracy and the MC predictor efficiency, along with various spatial interpolation filters. The experimental results show that for a block size of 16×16, quarter-pel resolution is sufficient for TV signals, and half-pel resolution is sufficient for video communication.

Although generations of video codecs have been targeting block sizes and MV accuracy for compression, the improvements from both paths might seem limited in terms of compression rate, i.e. no linear trends expected on increasing compression rate purely through refining MV accuracy. Furthermore, as H.264 employs block-sizes down to 4×4, further block re-sizing is also limited in future block-based video codecs. In other words, improving on compression efficiency requires a more advanced coding structure and MC/ME techniques in future video coding standards, possibly extending from the above mentioned two motion trends. In the remainder of this section, we discuss state-of-the-art ME techniques at a system level.

2.1.2.1. Block-Matching Motion Estimation

Among all motion estimation techniques, the block matching algorithm (BMA) is the main approach in existing video encoders. Each video frame is first partitioned into MBs. For each MB in P- and B-frames, the encoder performs a search in the reference frame(s) in the area around the position of the current MB. The search range is specified to limit the computational cost while allowing for a certain range of translational motion.

Using the criterion of the least prediction error, usually mean square error (MSE) or sum of absolute differences (SAD), the best two-dimensional displacement is estimated and referred to as the motion vector (MV).

While it is the most straightforward and intuitive approach for motion estimation, block matching is also the most widely used motion search strategy. A comparison of the target block is made with every possible reference block in a certain temporal and spatial range and a difference measure (e.g. mean square error) is computed for each of them. The displacement between the reference block associated with the smallest error measure is used as the MV for the target block. The several important parameters that are configurable in a block-matching search include the step size (how closely distributed the MVs are, e.g. 1 for pixel-wise ME, 16 for macroblock-wise ME), matching window size (the number of pixels in the neighbourhood that are compared for the error measure), the search range (the maximum possible MV) and MV accuracy.

The full-search block-matching strategy exhausts every pixel in every window that is within the search range. For example, to get an 8×8 block-wise quarter-pel MV field for a QCIF (176×144) frame, with the usual search range of 16×16, a window size of 16×16 and an MSE error criterion, 415M comparison operations, 415M summing operations and 415M multiplication operations — 16×16×64×64×22×18 = 415M — would be required for each frame, which is enormous. In order to overcome this computational drawback, a large amount of work has been done to design alternative fast search algorithms for block-matching, such as the 2D logarithmic search [38], the three-step search [39], the conjugate direction search [40], the cross search [41], the four-step search [42] and the block-based gradient descent search [43]. While these approaches attempt to shorten the search path on a square shaped search window, diamond [44] and hexagon [45] shaped search patterns have also been found to further speed up the job of searching. Some of these fast algorithms are so simple and effective that they have been recommended, although not specified, by the standards.

A hierarchical motion structure is one of the many techniques designed to reduce the computational load. Exploiting the fact that neighbouring motion vectors tend to be similar, the current and reference frames are filtered and sub-sampled to obtain two pyramids of frames, with the original full-size frames at the bottom, and their down-sized

versions on top in a descending order. Conventional blocking-matching is performed on a top level, resulting in a small MV field that can be stretched into a rough estimate of the larger MV field for the level beneath. This multi-resolution structure not only reduces ME computation, but also avoids local optima that may lead to false MVs.

2.1.2.2. Rate-Constrained ME in H.264

Sullivan *et*. *al*. first proposed rate-constrained ME in 1991 [46], which was later included in the H.263 and MPEG-4 standards [47]. Instead of only minimizing the measured error, rate-constrained ME obtains an MV that minimizes the weighted sum of the error and the bits required for coding the MV itself, as in Equation 2.1.

$$\mathbf{MV}' = \underset{\mathbf{MV}\in\mathbf{M}}{\operatorname{argmin}}\left(D_{DFD}(\mathbf{MV}) + \lambda_{MOTION} R_{MOTION}(\mathbf{MV})\right) \qquad \text{Equation 2.1}$$

where **M** is the set of motion vectors within the search range. The measured distortion associated with the MV is

$$D_{DFD}(\mathbf{MV}) = \sum_{(x,y)\in A} |s[x,y,k] - s'[x - MV^X, y - MV^Y, k-1]|^P \qquad \text{Equation 2.2}$$

where $P = 1$ for the sum of absolute differences (SAD) and $P = 2$ for the sum of squared differences (SSD), the choice of which is made through experimentation.

2.2. Global Motion Estimation

Global motion, in this work, is defined as the dominant motion in the observed motion field. In many cases, background makes up most of the frame. Hence, background motion, which is caused by camera motion, represents the global motion. However, in some cases, a large foreground moving object may occupy most of the frame, so in such cases, "global" motion actually represents the motion of this object.

Global motion can be useful in video coding, error concealment in video communications, and content-based video analysis, such as video object segmentation,

background modelling and video indexing [7][8][9]. However, unlike local motion (block MVs), global motion is generally not available in standard compressed video, but can be estimated in either pixel domain [2][4][5][8][49] or compressed domain [3][11][12][13][29]. In this section, we address the issue of estimating global motion using coarsely sampled (i.e. block-based) MV fields from compressed video. The motivation is to find an efficient GME approach in the compressed domain and significantly reduce the computational complexity of GME compared to pixel-based approaches.

As we discussed in Section 2.1, MVs in the compressed bitstream are often imperfect and inconsistent with real motion. In order to improve the estimation accuracy, an important part of most MV-based GME approaches is to remove MV outliers, i.e. those MVs that do not fit well into the global motion model.

2.2.1. Popular Pixel- and MV-based GME approaches

Global motion estimation can be broadly classified into pixel- and MV-based approaches. In general, the idea behind a pixel-based approach is to use the image luminance signal from a pair of frames to model the global motion, while the MV-based approach uses block-based MVs that are available in a compressed video stream. The MV-based approach has an advantage of lower computational complexity, which may come at a cost of lower accuracy.

2.2.1.1. Pixel-based GME approaches

Several pixel-based GME approaches are based on utilizing an image luminance signal to model global motion, where a Gauss-Newton gradient descent algorithm is commonly used to estimate motion parameters due to its fast convergence and reliable performance on minimizing the cost function.

Krutz *et*. *al*. proposed a pixel-based method that uses phase correlation to find an initial set of motion parameters [5], which initializes the motion model as a translational motion for the gradient descent algorithm. The described framework employs an image pyramid to reduce computational cost by downsampling the original image to a low resolution image and then starts a Gauss-Newton gradient descent on this low resolution image to obtain the first set of motion parameters. The converged

motion parameters are used as an initial parameter set for a new round of gradient descents on the higher resolution image. The process repeats until it reaches the original resolution of the image. A similar approach can also be found in [26], where authors proposed a simplified GME approach that only uses a downsampled image pair, rather than an image pyramid. The GME framework still adopts Gauss-Newton gradient descent to find motion parameters for an affine or perspective model.

Alzoubi and Pan developed a pixel-domain GME using fixed and random pixel subsampling patterns [10]. The idea is to use only a small subset of pixels to estimate global motion parameters, thus reduce the GME complexity. Their work focuses on finding a good set of pixels to obtain optimal parameter estimates. Their framework first uses a block matching method to find translational motion as a starting point, then selects a set of pixels with a pre-defined subsampling pattern, after which a Gauss-Newton gradient descent search iterates to find motion parameters that minimize the cost function.

2.2.1.2. Block-based GME approaches

The MV-based GME approach has been commonly used in many compressed video applications. The idea is to take an MV field extracted from a video stream and an MV field re-generated by a global motion model and obtain a set of global motion parameters to minimize the fitting error between these two fields. This approach alleviates the computational burden as MVs from compressed video are directly used. MVs represent local motion and can be used to initialize a global motion model without conducting motion estimation. In the literature, there are three main approaches in this category.

The first approach is to use an iterative procedure that applies gradient descent regression to the MVs to estimate model parameters, as proposed in [3], where the authors construct a framework to estimate global motion parameters by applying a gradient descent over a pyramid of motion models. The original MV field is first initialized from 2-parameter translational model and then it is used to initialize an 8-parameter perspective model by carrying out the gradient descent regression.

The second approach is similar to the first approach but using a least-squares solution. The idea is to use the MVs in an over-determined system of equations and use the least-squares solution to obtain global motion parameters by employing a pseudo-inverse [11][12]. As shown in [12], better performance can be obtained by introducing a weighting factor, the so called "M-Estimator," which is used to weight MVs according to their reliability, thereby performing outlier suppression.

The first and second approaches are also categorized as "iterative robust estimators," while the third popular approach is a so called RANdom SAmple Consensus (RANSAC) method [22]. RANSAC is a commonly used statistical method for outlier removal in various computer vision problems. It is an iterative method to filter out MV outliers by randomly choosing a set of MVs that best fits the motion model. The accuracy is generally increased as more iterations are carried out. It usually requires a large number of iterations to achieve reliable results.

2.2.2. Block-based GME and MV Outliers

The goal of MV-based GME is to infer the global motion parameters from the MVs; in the case of compressed video, MVs are available in the bit stream and do not need to be re-estimated. However, it is usually found that MV fields contain certain MVs (called outliers) that do not fit into a particular global motion model, thus for accurate MV-based GME, it is important to remove those outliers from the MV field. We distinguish two types of MV outliers:

Type-1) "Noisy" MVs, which are often found in areas where motion estimation fails to capture real motion, such as regions with little or no texture, boundary regions of a moving object, and regions with repetitive texture pattern. MVs associated with non-rigid or very small objects may also fall into this category;

Type-2) These outliers are the MVs from any object that does not fit the 8-parameter motion model of the background. Hence, there are two sources of Type-2 outliers: 1) objects that are really moving relative to the background, and 2) static objects that are not far enough from the camera to "blend into" the background and therefore do not fit the 8-parameter motion of the background. The common characteristic of Type-2 outliers is that they appear in spatially contiguous areas of the frame

occupied by the two kinds of objects mentioned above, and tend to be similar to their neighbours, i.e., they don't look "noisy" like Type-1 outliers.

It is important to remove both types of outliers in order to obtain accurate motion parameter estimates. In this chapter, we focus on developing a technique that effectively removes Type-1 MV outliers, and incorporating it into a block-based GME framework. A more general approach that deals with both Type-1 and Type-2 outliers is discussed in Chapter 4, where we further incorporate object segmentation into GME for simultaneously extracting object information and modelling global motion.

2.2.3. Characterization of the MV Field with a Moving Camera

The MV field in a compressed domain approximates the moving trajectory of each image unit (block or macro-block) between two frames, however, MVs generated by a conventional block-matching algorithm in video codecs are often imperfect and inconsistent with object and camera motion. To characterize the MV field with a moving camera, we use a 2D parametric global motion model to simulate camera motion.

In [3], four popular 2D motion models for global motion (translational, geometric, affine and perspective) have been summarized, with the 8-parameter perspective model being the most general. This model accurately represents the 2D projection of the 3D rigid motion of a planar surface [6]. Curved surfaces require higher-order models. However, since the background is usually far from the camera, it can be approximated by a planar surface [6], so in this work, we adopt the 8-parameter perspective model to represent the motion of the background. The perspective model is described by a vector of its parameters, $\mathbf{m} = [m_0, \cdots, m_7]$. Given $(x,\ y)$ and $(x',\ y')$ as the coordinates in the current and reference frame, respectively, the perspective transformation is defined as:

$$x' = \frac{m_0 x + m_1 y + m_2}{m_6 x + m_7 y + 1}, \qquad y' = \frac{m_3 x + m_4 y + m_5}{m_6 x + m_7 y + 1}, \qquad \text{Equation 2.3}$$

Defining $\mathbf{x} = [x, y, 1]^T$ and $\mathbf{x}' = \left[N_x, N_y, D\right]^T$, where N_x, and N_y are the numerators in Eq. 2.3 and D is the denominator, Eq. 2.3 can be represented by a homographic mapping:

$$\mathbf{x}' = \mathbf{H}(\mathbf{m}) \cdot \mathbf{x} = \begin{bmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ m_6 & m_7 & m_0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix},$$

$$x' = {}^{N_x}/_{D}, \qquad \text{Equation 2.4}$$

$$y' = {}^{N_y}/_{D},$$

The homographic mapping can also be decomposed into a product of matrices containing the focal lengths and rotation angles [9].

$$\mathbf{H}_{k,l}(\mathbf{m}) = \begin{bmatrix} f_k & 0 & 0 \\ 0 & f_k & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{R}_z \cdot \mathbf{R}_x \cdot \mathbf{R}_y \cdot \begin{bmatrix} f_l^{-1} & 0 & 0 \\ 0 & f_l^{-1} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad \text{Equation 2.5}$$

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi_x & -\sin\varphi_x \\ 0 & \sin\varphi_x & \cos\varphi_x \end{bmatrix},$$

$$\mathbf{R}_y = \begin{bmatrix} \cos\varphi_y & 0 & \sin\varphi_y \\ 0 & 1 & 0 \\ -\sin\varphi_y & 0 & \cos\varphi_y \end{bmatrix}, \qquad \text{Equation 2.6}$$

$$\mathbf{R}_z = \begin{bmatrix} \cos\varphi_z & -\sin\varphi_z & 0 \\ \sin\varphi_z & \cos\varphi_z & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

Here, f_k and f_l denote the camera focal length of the k-th and l-th frame, respectively, and φ_x, φ_y, and φ_z, denote the rotation angles around *x*-, *y*-, and *z*-axes. In this model, the X- and Y- components of the MV field at (x, y) in the current frame are given by:

$$MV^X(x, y; \mathbf{m}) = x' - x$$

$$MV^Y(x, y; \mathbf{m}) = y' - y \qquad \text{Equation 2.7}$$

MVs that come from a given motion model (a given vector of motion parameters $\mathbf{m}$) usually exhibit fairly strong spatial correlation. We use this observation to score the MV reliability, and check the likeliness of MVs fitting in a motion model. In order to estimate how similar or how different the neighbouring MVs from a given model could be, we performed the following experiment. We used a range of camera parameters in the homographic mapping that are commonly found in practice [24] [25]: a focal length of 200 – 1000; and a focal length change ratio between consecutive frames of 0.9 – 1; an angular velocity of [−1.6, 1.6] degrees per frame for *x*-, *y*- and *z*-axes.

We created 14,000 combinations of camera parameters (focal length, rotation angle, and translational distance) from these ranges and, for each combination, we synthesized an MV field in floating-point MV precision according to Eq. 2.5 and 2.6. We then measured the average relative difference in magnitude (D_{mag}) and phase (D_{ph}) between each MV and its eight surrounding neighbours (hereafter referred to as the "8-neighbourhood"). The histograms of these quantities were then created, and the 90-th percentile was computed. Assuming a CIF (352×288) resolution video, Table 2.1 lists the 90-th percentile values of D_{mag} and D_{ph} for block sizes of 4×4, 8×8, 16×16, and 32×32 pixels.

Table 2.1: The 90-th percentile of magnitude and phase difference

Block Size	D_{mag}	D_{ph} (Degrees)
32×32	1.0	45
16×16	0.4	19
8×8	0.2	9
4×4	0.1	4

The values of $D_{mag}$and D_{ph} for various combinations of settings are shown in Fig. 2.3. The 90-th percentile for $D_{mag}$was 0.4, and the 90-th percentile for $D_{ph}$was 19 degrees, meaning that 90% of the MVs in a MV field described by a perspective model, with the camera parameters from the ranges listed above, have the average relative magnitude difference from their neighbours of 0.4 or less, and an average phase difference of 19 degrees or less. These 90-th percentile values are used in the next section to identify MV outliers and set the thresholds for outlier rejection.

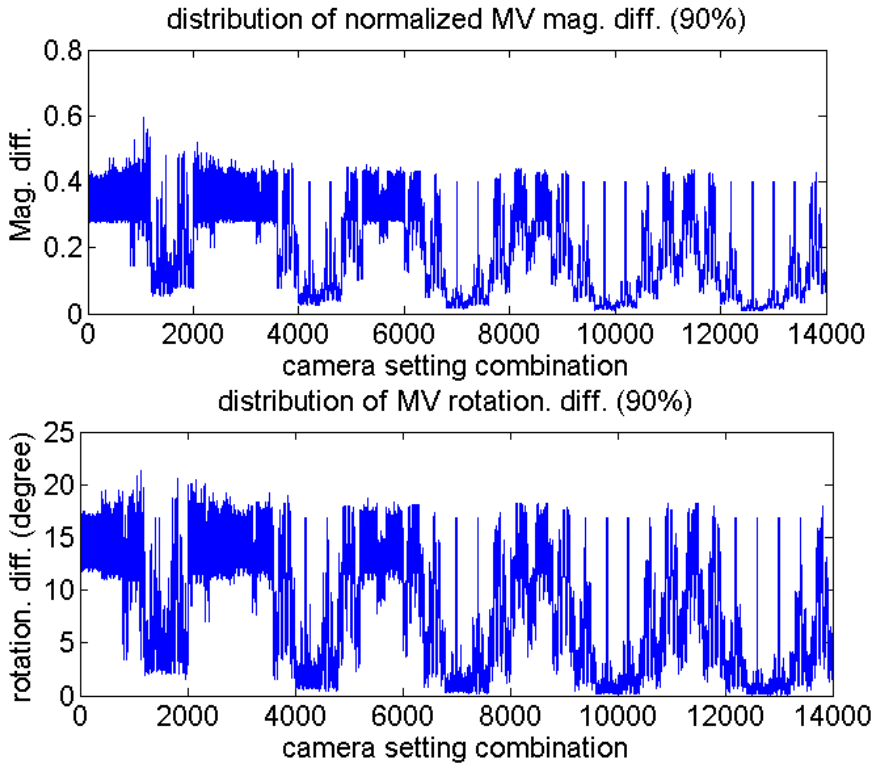


Figure 2.3: The values of average MV magnitude and rotation difference for various settings

2.2.4. MV Outlier Identification and Rejection Cascade for Global Motion Estimation

MV outliers in this section are defined as MVs that do not fit into a global motion model, i.e. Type-1 and Type-2 outliers described in Section 2.2.2. Identifying and removing outliers from an MV field plays an important role in global motion segmentation and content based video applications. In this section, we propose using a cascade-of-rejectors approach to remove outliers from an input MV field.

The cascade-of-rejectors approach has been very successful in fast object detection [48], where it is used to quickly verify the presence or absence of certain object features. We propose a similar cascade approach to identify and remove MV outliers, which consists of three filters, as shown in Fig. 2.4. The input MV field is subject to

testing in the first filter, then the MVs declared as inliers are further tested in the second filter, and so on.

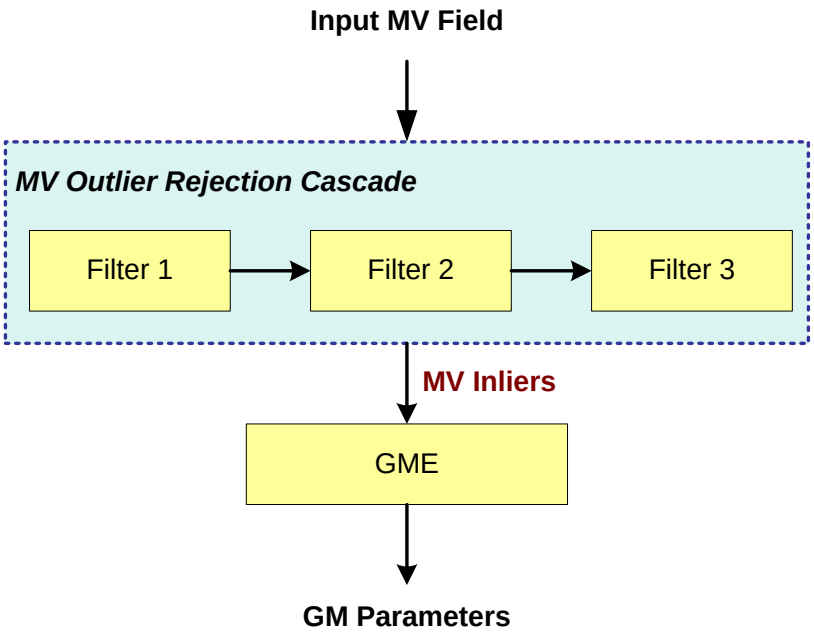


Figure 2.4: The proposed MV outlier removal cascade.

To test each input MV, the filters in the cascade employ the following strategy. Let $\mathbf{MV}_i$ be the input MV to be tested in filter j, where $j \in \{1,2,3\}$. Associated with $\mathbf{MV}_i$is the set S_i^j of MVs computed from the 8-neighbourhood of $\mathbf{MV}_i$ as shown in Fig. 2.5, where the location of $\mathbf{MV}_i$is shown in grey. For filter 1, S_i^1 consists of individual MVs from the neighbourhood of $\mathbf{MV}_i$, as shown in Fig. 2.5(a). For filter 2, S_i^2 consists of the averages of diagonally opposite MVs from the neighbourhood of $\mathbf{MV}_i$, as shown in Fig. 2.5(b). Finally, for filter 3, S_i^3 consists of the averages of triangularly opposite MVs from the neighbourhood of $\mathbf{MV}_i$, as shown in Fig. 2.5(c). There are at most eight MVs in S_i^1, and at most four MVs in each of S_i^2 and S_i^3.

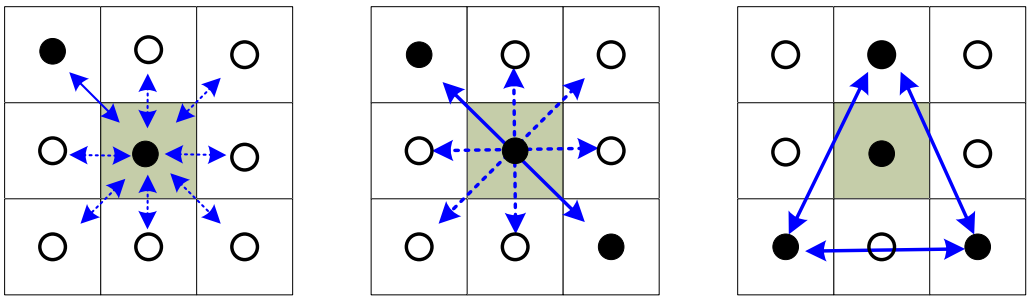


Figure 2.5: The construction of MVs in S_i^j, (a): S_i^1, (b): S_i^2, and (c): S_i^3.

Once S_i^j is constructed, we test the following conditions for each $\mathbf{MV}_k \in S_i^j$, and count how many times the following conditions are satisfied:

$$\frac{\|\mathbf{MV}_i\| - \|\mathbf{MV}_k\|}{\|\mathbf{MV}_i\|} < T_{mag}^j, \qquad \text{Equation 2.8}$$

$$|\varphi(\mathbf{MV}_i) - \varphi(\mathbf{MV}_k)| \leq T_{ph}^j, \qquad \text{Equation 2.9}$$

where T_{mag}^j and T_{ph}^j are the thresholds for maximum relative magnitude difference, and maximum phase difference, respectively. To avoid the computation of phase $\varphi(\cdot)$, Equation 2.9 can be rewritten as:

$$\langle \mathbf{MV}_i, \mathbf{MV}_k \rangle > \|\mathbf{MV}_i\| \cdot \|\mathbf{MV}_k\| \cos\left(T_{ph}^j\right) \qquad \text{Equation 2.10}$$

Let N_i^j be the number of times the above conditions are satisfied. Note that $N_i^1 \leq 16$, and $N_i^j \leq 8$ for $j \in \{2,3\}$. The weighted count is given by

$$WN_i^j = W_i^{j-1} \cdot N_i^j, \qquad \text{Equation 2.11}$$

where

$$W_i^j = \exp\left(-\left(WN_{max}^j - WN_i^j\right)\right), \qquad \text{Equation 2.12}$$

$WN_{max}^j = max_j\left(WN_i^j\right)$ and $W_i^0 = 1$ for all i. The weight W_i^j is a measure of how similar $\mathbf{MV}_i$is to vectors in S_i^j.

Table 2.2: Magnitude and phase thresholds

Filter *j*	T_{mag}^j		T_{ph}^j (Degrees)	
	16×16	8×8		16×16
1	0.4	0.2	1	0.4
2	0.2	0.1	2	0.2
3	0.1	0.05	3	0.1

The magnitude and phase thresholds in Equation 2.8 – 2.9 are determined based on the 90-th percentile values for the relative magnitude and phase difference found in Section 2.2.3. The thresholds for MV field with 16×16 and 8×8 blocks are set as shown in Table 2.2. For filter 1, the thresholds are equal to the 90-th percentile values from Table 2.1. These thresholds are halved for filter 2, and halved again for filter 3.

Each of the three filters in the cascade is set to keep the same fraction of inliers in order to satisfy the target fraction of inliers. If $p \in [0,1]$ is the fraction of inliers we want from the cascade, then each filter is set to keep $p^{1/3}$ of the input MVs, and remove the rest as outliers. For example, if we want to keep 70% of MVs as inliers, then $p = 0.7$, $p^{1/3} \approx 0.888$, so each filter will keep approximately 88.8% of its input MVs as inliers, and remove 11.2% as outliers. The filtering operation is summarized below:

1. Symmetrically extend the MV field across frame boundaries, flag all MVs as inliers and set $j = 1$.

2. For each inlier $\mathbf{MV}_i$, find the weighted count W_i^j. Note that previously declared outlier MVs are included in the neighbourhoods (S_i^j) of inlier MVs.

3. Sort MVs in descending order of their weighted counts.

4. Declare the target number of MVs at the bottom of the sorted list as outliers.

5. If $j = 3$, then stop. Otherwise, set $j = j + 1$, and move on to the next filter, repeating steps 2 – 5.

2.2.5. Global Motion Estimation with MV Outlier Removal

In our investigation on MV reliability [28], we found that MV outlier filtering, such as the filter proposed in [23], is computationally effective and works very well on Type-1 outliers, since these "noisy" MVs tend to be different from their neighbouring MVs, but it may run into problems when faced with Type-2 outliers. RANSAC tends to work well on Type-2 outliers, but its performance is not as good on Type-1 outliers. Meanwhile, iterative robust estimators tend to work well on Type-1 outliers and also on Type-2 outliers, as long as their percentage in the MV field is not large. These findings were used as a basis for developing an effective, yet accurate, GME approach that employs the outlier removal strategy developed in the previous section.

In most cases, Type-1 MV outliers can be reliably detected by checking their similarity to their neighbouring MVs. In this section, we use the filter cascade from the previous section to remove MV outliers prior to GME [28]. The cascade examines the magnitude and phase difference between an MV and its neighbours and then removes a prescribed fraction of worst-fitting MVs from the MV field as outliers. In our experiments this fraction is set to be 0.15.

We evaluate and compare this proposed outlier rejection approach in the context of GME with three state-of-the-art methods: the iterative Gradient Descent (GD) approach [3], the least squares solution using an M-Estimator (LSS-ME) [11], and RANdom SAmple Consensus (RANSAC) [22]. We chose GD as the platform to examine the effects of using the proposed MV outlier removal cascade to pre-process the MV field (CAS_GD). We compare MV outlier rejection capabilities of our cascade against the filter from [23] (FLT_GD). We also implemented LSS-ME [11] and RANSAC with least-square regression (RAN_LS) [22] for further performance comparison. Parameter C in LSS-ME is set to 2 during evaluation.

We first evaluate the algorithms on a synthetic MV field. We use four sets of motion parameters as in [3], shown in Table 2.3, to synthesize the test MV fields. These models also fall within the range of parameters used in Sections 2.2.3 to set the filter

thresholds. After the MV field is synthesized (assuming a CIF resolution and 16×16 blocks), as in [3], the MVs are corrupted by independent zero-mean Gaussian noise in both *X*- and *Y*- components. Following that, outlier MVs (groups of connected MVs pointing in a random direction) are added to simulate foreground moving objects. As in [3], the performance criterion is the signal-to-noise ratio (SNR) between the MV field generated by parameters $\mathbf{m}$ (Table 2.3) and the MV field generated by the estimated parameters $\hat{\mathbf{m}}$.

Table 2.3: Global motion parameters used for testing

Model	Motion parameters
GM 1	$\mathbf{m}$=[0.9, 0, 10.4238, 0, 0.95, 5.7927, 0, 0]
GM 2	$\mathbf{m}$=[0.9964, –0.0249, 1.0981, 0.0856, 0.9457, –7.2, 0, 0]
GM 3	$\mathbf{m}$=[0.9964, –0.0249, 6.0981, 0.0249, 0.9964, 2.5109, -2.7×10^{-5}, 1.9×10^{-5}]
GM 4	$\mathbf{m}$=[1, 0, 4.4154, 0, 1, 0, -1.13×10^{-4}, 0]

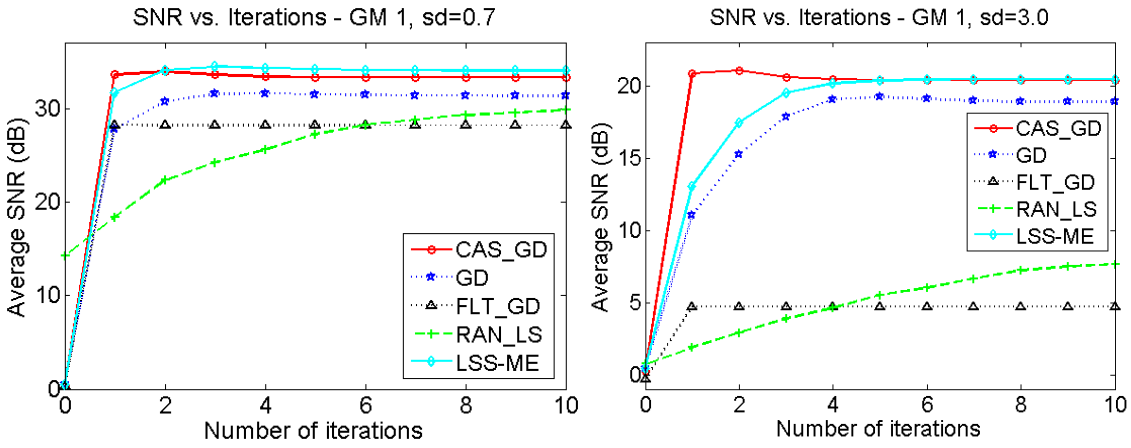


Figure 2.6: SNR vs. Iterations with a synthetic MV field (GM-1) corrupted by Gaussian noise with $\sigma \in \{0.7, 3.0\}$.

Figure 2.6 shows the results of five GME approaches in terms of SNR vs. the number of GME iterations. In this experiment, the MV field (generated using parameters from GM 1) is only corrupted by noise. Simulation results with $\sigma \in \{0.7,3.0\}$ are shown in

Fig. 2.6 (the results with $\sigma \in \{1.5, 2.2\}$, as well as results with other models from Table 2.3, follow similar trends). Each result is averaged over 50 runs. The filters in our cascade were set to give 70% of inliers overall in order to facilitate a fair comparison with the results in [3]. We observed that the CAS_GD converges faster than GD, LSS-ME and RANSAC and achieves a very close SNR to LSS-ME. Both CAS_GD and LSS-ME have a higher SNR than other methods, especially as the noise variance increases. In Table 2.4, we list the converged SNR values of these four methods for all four GM models from Table 2.3. FLT_GD yields the worst performance among the tested methods because the filter from [6] usually removes too many input MVs.

Table 2.4: SNR in the MV field (dB), corrupted only by Gaussian noise

GM Model	GME Algorithms	Standard Deviation of Gaussian Noise			
		0.7	1.5	2.2	3.0
GM 1	**CAS_GD**	**33.98**	**27.60**	**23.87**	**21.50**
	GD	31.71	24.63	21.85	20.11
	FLT_GD	28.15	15.83	11.19	8.73
	RAN_LS	29.61	17.60	13.58	9.08
	LSS-ME	34.23	27.79	23.47	20.56
GM 2	**CAS_GD**	**37.28**	**31.28**	**27.92**	**25.11**
	GD	35.02	29.01	26.33	23.01
	FLT_GD	33.92	23.04	17.37	13.62
	RAN_LS	31.57	20.87	17.28	14.20
	LSS-ME	38.11	31.12	27.39	24.65
GM 3	**CAS_GD**	**33.65**	**27.16**	**23.05**	**21.02**
	GD	33.01	26.53	23.73	21.18
	FLT_GD	30.49	18.43	12.23	9.77
	RAN_LS	29.38	18.18	14.51	12.39
	LSS-ME	34.64	29.05	25.51	21.55
GM 4	**CAS_GD**	**37.67**	**30.64**	**26.39**	**23.19**
	GD	35.56	29.11	26.22	23.14
	FLT_GD	34.51	23.27	17.29	13.53
	RAN_LS	33.72	21.27	17.78	14.26
	LSS-ME	38.11	31.48	28.11	24.23

Table 2.5: SNR in the MV field (dB), corrupted by both noise and outliers

GM Model	GME Algorithms	Outlier Percentage ($\sigma = 1.5$)			
		0 %	2 %	10 %	20 %
GM 1	**CAS_GD**	**27.56**	**16.16**	**8.74**	**5.35**
	GD	24.75	15.81	8.57	4.40
	FLT_GD	15.46	12.97	7.97	3.58
	RAN_LS	17.13	13.44	7.94	4.68
	LSS-ME	27.67	15.93	8.52	4.42
GM 2	**CAS_GD**	**31.24**	**19.59**	**11.64**	**6.90**
	GD	29.31	19.38	11.59	6.46
	FLT_GD	23.05	17.82	11.25	6.65
	RAN_LS	20.34	16.69	11.01	6.32
	LSS-ME	31.30	19.47	11.49	6.47
GM 3	**CAS_GD**	**26.81**	**17.35**	**10.38**	**7.19**
	GD	27.04	17.21	10.12	6.15
	FLT_GD	17.54	13.61	8.20	3.11
	RAN_LS	18.14	14.96	9.66	5.90
	LSS-ME	28.32	17.31	10.04	6.17
GM 4	**CAS_GD**	**30.06**	**20.13**	**12.66**	**8.93**
	GD	29.11	19.93	12.43	7.93
	FLT_GD	23.20	18.99	11.81	6.01
	RAN_LS	21.10	17.37	11.97	7.77
	LSS-ME	31.24	20.06	12.39	7.79

Next, we corrupted the MV fields by both noise ($\sigma = 1.5$) and outliers made up of connected regions of 3×3, 6×6 and 9×9 MVs in the center of the frame (2%, 10% and 20% of the total MV field size). These MV outliers are generated by adding the vector $(5,5)$ to the MVs generated by the global motion model. The results for GM 3 are shown in Fig. 2.7, where we can observe that the performance of RANSAC improves compared to other methods as the percentage of MV outliers increases. Average (converged) SNRs for all four GMs are listed in Table 2.5, while the number of iterations needed to achieve SNRs from Tables 2.4 and 2.5 are listed in Table 2.6.

Table 2.6: Average number of iterations to achieve performance above

GME Algorithms	Noise only (No outliers) $\sigma \in \{0.7, 1.5, 2.2, 3.0\}$				Outliers and noise ($\sigma = 1.5$)		
	0.7	1.5	2.2	3.0	2%	10%	20%
CAS_GD	2	2	2	2	2	2	2
GD	6	6	6	6	6	6	6
FLT_GD	2	2	2	2	2	2	2
RAN_LS	14	132	>500	>500	144	199	327
LSS-ME	4	4	4	4	4	4	4

Table 2.7: Global motion compensation performance, PSNR in dB

Sequences	CAS_GD	FLT_GD	GD	RAN_LS	LSS-ME
Flower Garden	22.19	21.44	22.30	21.87	22.48
Stefan	24.60	22.16	24.51	24.74	24.60
City	29.48	29.25	28.70	29.62	29.88
Tempete	27.83	24.98	26.51	27.86	27.66
Waterfall	34.86	24.25	34.71	35.48	34.69
Mobile Calenda	23.47	22.69	23.91	24.72	24.97
Coastguard	26.78	26.90	26.54	26.97	26.82
Average	**27.03**	**24.52**	**26.73**	**27.32**	**27.30**

We then evaluate algorithms on MV fields from real test sequences. Here test sequences are carefully selected to contain mostly camera motion. We use GME on the MVs to estimate the model and perform global motion compensation by warping the target frame onto the reference frame plane according to the model using bilinear interpolation [23]. If the sequence indeed contains only camera motion, and if GME is accurate, we should expect the frames compensated by global motion to be very close to the original frames. The similarity can be measured using the conventional Peak Signal-to-Noise Ratio (PSNR). We performed this experiment on seven test sequences listed in Table 2.7. Exhaustive search on 8×8 blocks is used to estimate the MVs prior to

GME. We compared the same five GME methods as in the previous section. This time, CAS_GD and FLT_GD incorporate only a single iteration of GD; plain GD uses six iterations while LSS-ME is set to use three iterations. The thresholds for 8×8 blocks listed in Table 2.2 are used in our cascade. The total processing time per frame was measured in MATLAB on a standard desktop PC with an Intel Pentium CPU at 3.0GHz and 2GB of RAM. This processing time includes all filtering and GME operations.

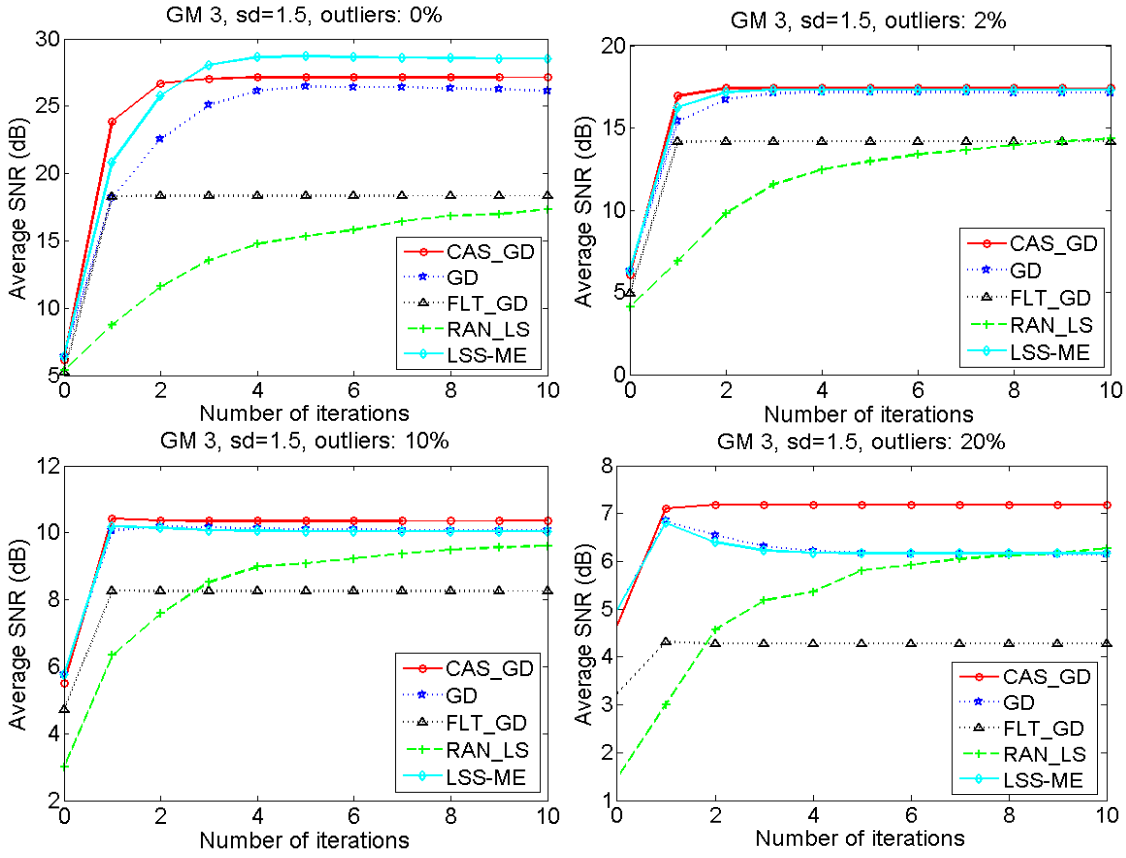


Figure 2.7: SNR vs. Iterations with synthetic MV field (GM 3) corrupted by Gaussian noise ($\sigma = 1.5$) and outliers {0%, 2%, 10%, 20%}.

Table 2.8 lists the average PSNR in dB and the average processing time on the seven test sequences. FLT_GD yields the fastest performance (36% faster than our CAS_GD), but also has the lowest PSNR performance (about 2.5dB worse than our

CAS_GD). Its PSNR performance, compared to other methods, seems to be especially poor on sequences with small camera motion, such as *Waterfall*, where its thresholding strategy removes too many MVs. Our CAS_GD is the next fastest method and achieves better PSNR (by about 0.3dB) than plain GD, simultaneously with a 73% speedup compared to plain GD. Finally, RAN_LS and LSS-ME give the best PSNR (both about 0.3dB higher than our CAS_GD), but at a significantly higher computational cost. Overall, our CAS_GD gives the best trade-off between accuracy and complexity. These results should be taken with a grain of salt, though, because the motion present in these sequences is not entirely due to camera motion. Nonetheless, the results provide some insight into the GME performance that can be expected on real sequences.

Table 2.8: Global motion compensation processing time, time in milliseconds

Sequences	CAS_GD	FLT_GD	GD	RAN_LS	LSS-ME
Flower Garden	25.3	16.9	43.1	137.6	298.6
Stefan	25.0	16.2	42.2	179.2	442.0
City	25.1	15.9	44.0	143.2	479.3
Tempete	24.9	15.8	42.5	98.9	488.2
Waterfall	23.9	15.0	42.7	97.1	478.5
Mobile Calenda	24.8	15.3	42.2	117.4	477.5
Coastguard	24.8	16.2	44.2	117.7	464.6
Average	**24.8**	**15.9**	**43.0**	**127.3**	**446.9**

3. Object Segmentation from Compressed Video

Object segmentation is a useful technique for object-based video representation and description [76] [77]. In recent years, object segmentation and tracking has gained more attention due to an increasing number of content-based multimedia applications. Extraction of moving regions in compressed video can benefit many multimedia applications, such as video indexing and retrieval [78], image querying [50], video database browsing [51], and predictive decoding for interactive video applications [52]. By employing segmentation to automatically discriminate the moving regions, these applications can classify the objects at various levels of detail, and achieve different system goals by tracking the particular object and analyzing its trajectory.

This Chapter will present multiple proposals on extracting moving objects from compressed video, starting with simple *k*-means clustering, which is further extended to employ a MV integration to refine object boundaries. As segmented boundary resolution is limited by the block-based MV field, we further incorporate spatial information into segmentation process to develop a coarse-to-fine object segmentation framework. Finally, a Markov Random Field (MRF) based segmentation approach is proposed to improve segmentation consistency across video frames.

3.1. Compressed-Domain Segmentation using *K*-means Clustering

3.1.1. Pixel-Domain vs. Compressed-Domain Segmentation

Depending on the domain in which processing is done, segmentation approaches can also be classified as: 1) segmentation in pixel domain [14][15][66] [79], which relies on image features (e.g., colour, edge and texture); 2) segmentation in a compressed domain [20][68][80][27][51][61][67-71], which only utilizes compressed video data (e.g., MVs and/or DCT coefficients); and 3) joint segmentation, which

performs part of the processing in a compressed domain, and the rest in the pixel domain [18][27][81].

Pixel-domain methods extract objects by exploiting visual features such as shape, colour and texture. In this case, the compressed video has to be fully decoded prior to segmentation. The high computational load and over-segmentation of possible moving objects are two major drawbacks of these methods.

Since video data is commonly stored in compressed form, compressed-domain segmentation appears more attractive for some applications. Compressed-domain methods exploit compressed domain data, such as MVs and DCT coefficients, to facilitate segmentation. These methods have significantly lower complexity than pixel-domain methods. In this section, we propose a compressed-domain segmentation approach, which takes MVs from the compressed video stream and uses *k*-means clustering to extract moving regions. This approach is also present in our previous work [93].

3.1.2. MV-Based Segmentation Using K-means Clustering

Motion information is available in the compressed video bit stream in the form of MVs. By isolating the regions of consistent motion, we can determine where the moving regions are in the frame. Figure 3.1 illustrates the idea of segmenting a moving region out of a frame based on motion consistency. Note that, since MVs are block-based, the region boundary will also follow block boundaries.

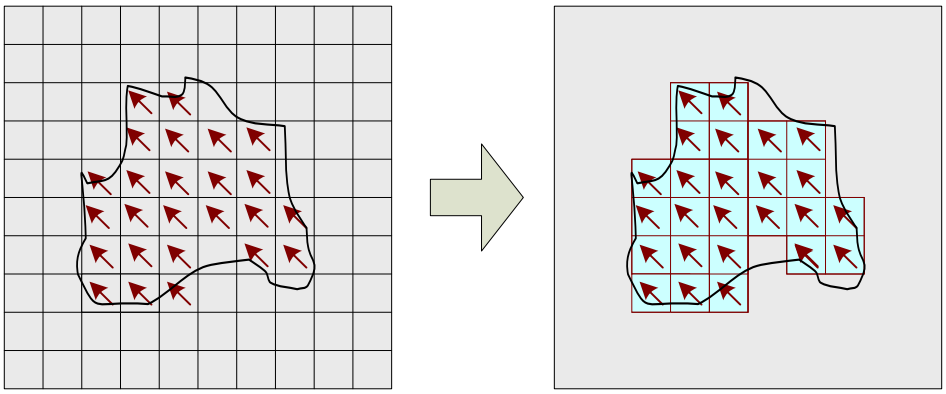


Figure 3.1: Illustration of block-based motion segmentation.

References [61] and [62] described methods for achieving robust motion segmentation with adaptive *k*-means clustering. For the purpose of identifying moving regions, we adopt a similar approach by combining *k*-means clustering [63] and motion consistency verification [64]. The flow diagram of this motion segmentation approach is shown in Fig. 3.2.

Motion information associated with each block is first extracted from the encoded bit stream. For intra-coded blocks, which do not have an associated MV, we assign a median MV computed from the 3×3 neighbouring blocks. The MVs are then normalized according to the temporal distance and direction indicated by the reference frame index [65] and mapped to the minimum block size supported by the particular coder — in our case 8×8 blocks since we are dealing with MPEG-4. This is done simply by splitting the larger blocks to the sub-blocks of the smallest supported block size, and assigning to each of the sub-blocks the MV of the parent block. The neighbouring MVs are then iteratively grouped into clusters using the *k*-means clustering algorithm with the help of the motion consistency model. The segmentation is carried out as follows:

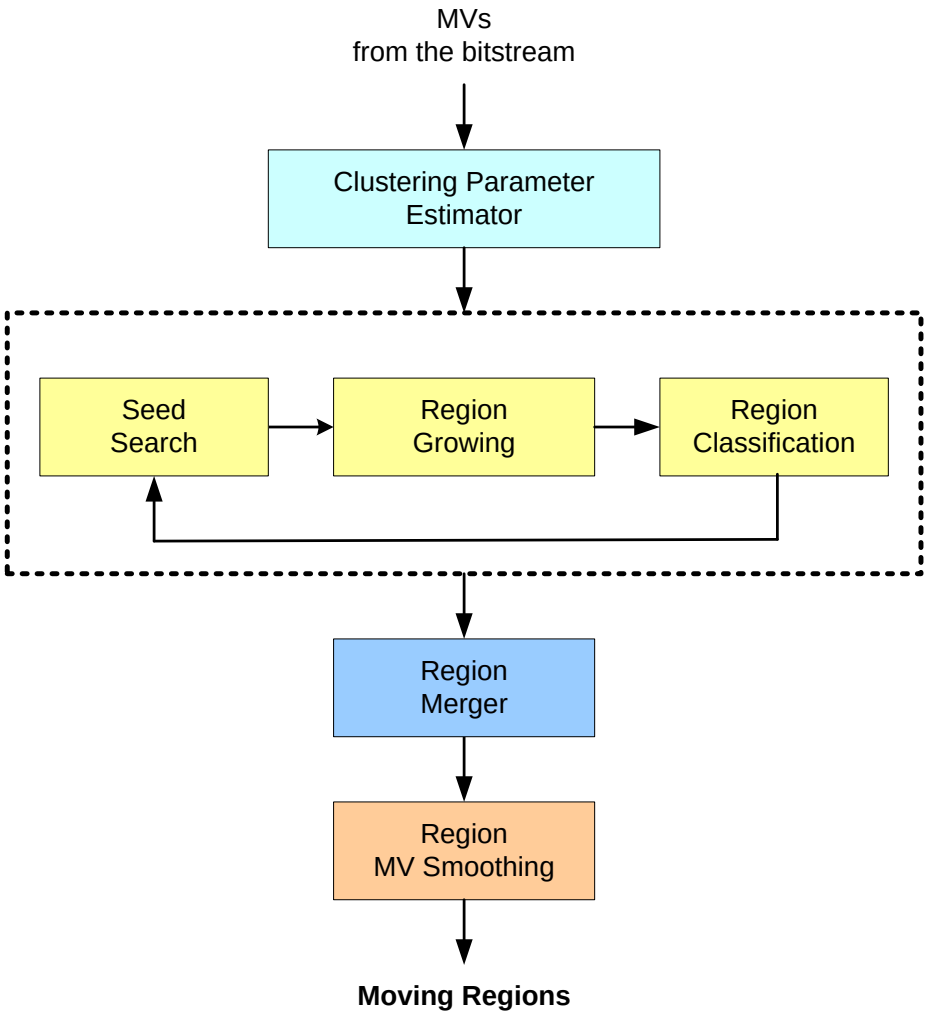


Figure 3.2: The proposed *k*-means motion segmentation in compressed domain.

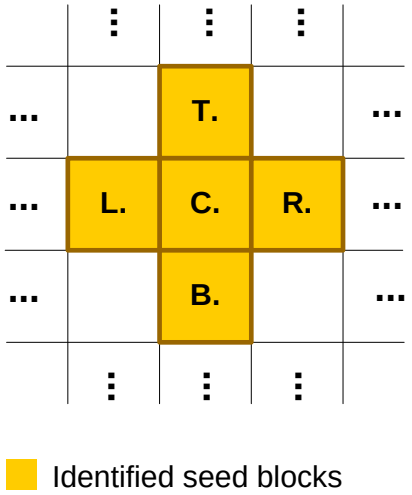


Figure 3.3: The seed pattern.

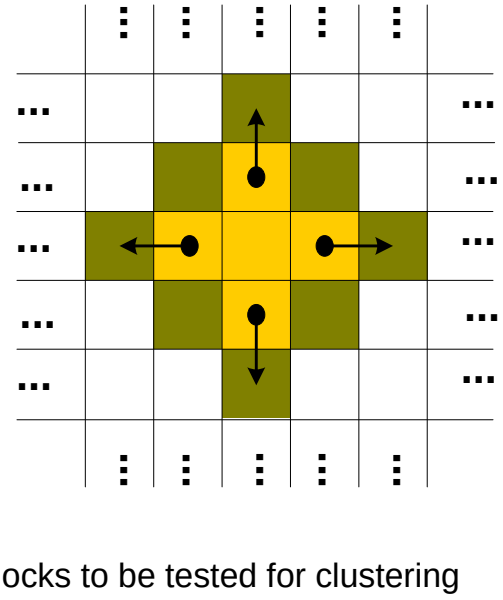


Figure 3.4: Region growing.

Step 1. MV statistics are computed first, to estimate the overall motion intensity in the clustering parameter estimator in Fig. 3.2:

$$\boldsymbol{\mu}_{\mathbf{MV}} = \frac{1}{N}\left(\sum_{i=1}^{N} \mathbf{MV}_i\right) \qquad \text{Equation 3.1}$$

$$\sigma_{\mathbf{MV}}^2 = \frac{1}{N} \cdot \sum_{i=1}^{N} \|\mathbf{MV}_i - \boldsymbol{\mu}_{\mathrm{MV}}\|^2 \qquad \text{Equation 3.2}$$

where N denotes the number of inter-coded blocks in the frame to be segmented, and i goes through all inter-coded blocks. Based on these statistics, we set two parameters: the region growing offset $D_{off} = \min(\sigma_{\mathbf{mv}}, 8)$, and the minimum moving region distance $D_m = \min(\sigma_{\mathbf{mv}}, 8)$. These two parameters are used in region growing and region merger, respectively.

Step 2. With the pattern from [64] shown in Fig. 3.3, a group of blocks with the most consistent MVs is chosen as the starting point ("seed") for region growing. The consistency of MVs in the pattern in Fig. 3.3 is measured by the total deviation of the MVs from the centroid MV. Let *Seed* = {Center, Top, Bottom, Left, Right} be the set of locations of the blocks in the seed pattern, and let $\mathbf{MV}_{cent}^{R}$ be the centroid MV of *Seed*. The total deviation of MVs in the *Seed* from the centroid is given by

$$D_{Seed} = \sum_{j \in Seed} \left\|\mathbf{MV}_{cent}^{Seed} - \mathbf{MV}_j\right\| \qquad \text{Equation 3.3}$$

We go through the entire frame to find the seed with minimum D_{Seed}. After we identify the seed, neighbouring blocks are tested for clustering, as shown in Fig. 3.4.

Step 3. With D_{off} from step 1) and the seed from step 2), a region will gradually be grown by clustering bordering blocks into the region if their motion is sufficiently similar to the prevalent motion inside the region. Let R be the set of locations of MVs in the current region (at the start of region growing, $R = Seed$). Motion consistency is

measured as the deviation of an MV from the centroid MV of region R, with the allowable deviation given by the following threshold:

$$D_{TH}^{R} = D^{R} + D_{off} \qquad \text{Equation 3.4}$$

where

$$D^{R} = \sum_{j \in R} \left\| \mathbf{MV}_{cent}^{Seed} - \mathbf{MV}_{j} \right\| \qquad \text{Equation 3.5}$$

and $\mathbf{MV}_{cent}^{Seed}$ is the centroid MV of region R. Now let $\mathbf{MV}_{i}$ be the MV at location i that borders the current region R. The region growing process checks whether this MV is sufficiently similar to the prevalent motion inside the region, by checking if the following condition is satisfied:

$$D_{i}^{R} = \left\| \mathbf{MV}_{cent}^{Seed} - \mathbf{MV}_{i} \right\| \leq D_{TH}^{R} \qquad \text{Equation 3.6}$$

If Eq. 3.6 is satisfied, we group $\mathbf{MV}_{i}$ into region R, otherwise $\mathbf{MV}_{i}$ is left ungrouped.

Step 4. If $\mathbf{MV}_{i}$ was grouped into region R, we update the MV centroid and the motion consistency parameters in Eq. 3.4 and 3.5 using the newly added $\mathbf{MV}_{i}$. Steps 3) and 4) are repeated until no more MVs can be grouped into region R, after which we start looking for the new seed.

Step 5. Repeat steps 2) to 4) until no further seeds can be found. Any remaining ungrouped MVs are grouped into the neighbouring region with the closest centroid to it, as measured in Eq. 3.6.

Step 6. Calculate the distance between centroid MVs of adjacent regions, and merge two adjacent regions if the distance between their centroid MVs is less than D_{m} , which was computed in step 1).

Step 7. Perform MV smoothing via vector median filtering with a 3×3 kernel size [20] to reduce MV noise inside each region. Only MVs from a given region are used in vector median filtering for smoothing the motion in that region. This way, we prevent the motion of one region (e.g., background) contaminating the motion in another region (e.g., moving object).

Figure 3.5 shows the results of the just described block-based motion segmentation with sequences *Flower Garden*, *Table Tennis*, and *Football*, where the original video frames are also displayed to show what the exact moving regions look like. Frame #2 from *Flower Garden*, Frame #20 from *Table Tennis*, and Frame #35 from *Football* are used for this example. To display the result of motion segmentation, moving regions are filled with different luminance values to distinguish them from each other.

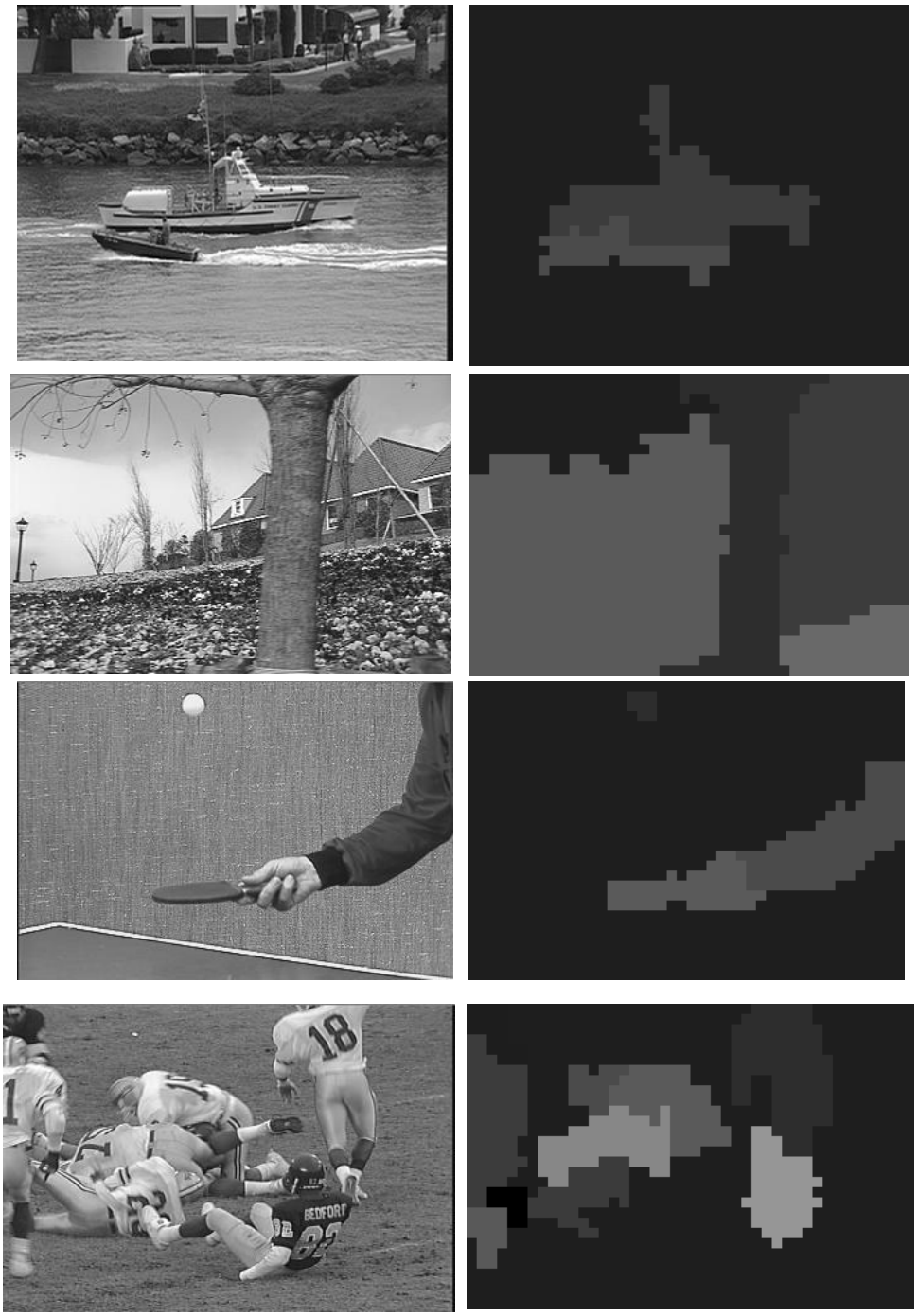


Figure 3.5: Block-based motion segmentation results. [Top to Bottom]*Coastguard*, *Flower Garden*, *Table Tennis* and *Football*.

From these results, we can see that the *k*-means based motion segmentation algorithm segments the moving regions reasonably well at the block-precision level. However, the true boundaries of moving regions do not necessarily match the block-based boundaries obtained by motion segmentation. Therefore, boundary refinement is necessary if a smooth boundary is expected.

To address the boundary refinement problem, we made two attempts in this thesis: 1) create a dense MV field by MV integration and conduct the refinement completely in compressed domain, and 2) incorporate spatial information into segmentation process by decoding the bitstrem and reconstructing video frames, and then refining the object boundary in pixel domain. The details of these two approaches are present in Sections 3.3 and 3.4, respectively, along with performance evaluations.

3.2. Motion Segmentation Using Motion Integration

Segmentation methods, which operate directly on a sparse (block-based) MV field [67-68], have low complexity, but often suffer from poor localization of object boundaries, and inconsistency in the number of segmented regions from frame to frame. Alternatively, one can create a dense (pixel-based) MV field by interpolation and then run segmentation on the dense field, at the cost of significantly higher complexity [61][69-70]. It remains a challenge to devise a fully compressed-domain segmentation method that achieves accurate object boundaries at reasonable computational expenses.

In this section, we present a novel compressed-domain segmentation approach that uses only block-based MVs, and yet achieves moving region boundaries with pixel precision. Some of the material in this section can also be found in our earlier work [124]. A system diagram of the proposed segmentation is shown in Fig. 3.6. First, MVs are used for coarse segmentation at block-precision. Second, boundary regions are identified, and the motion field density is increased inside the boundary regions by performing pixel-based MV integration over a set of video frames. Finally, contour evolution is employed to refine moving region boundaries.

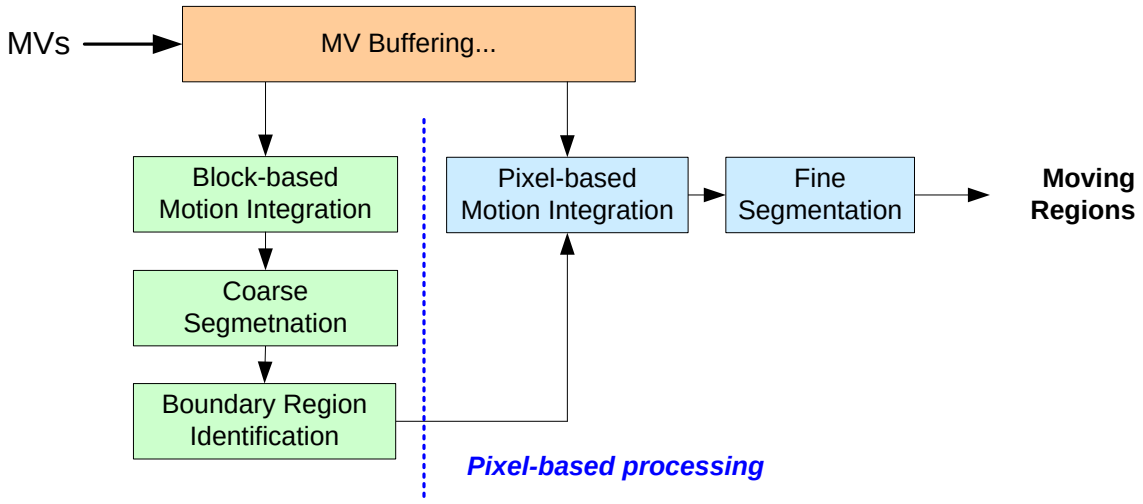


Figure 3.6: System diagram of moving region segmentation.

3.2.1. Motion Vector Integration

MVs in the compressed bitstream are usually generated using block-based motion estimation at the encoder by minimizing prediction error rather than finding true moving trajectories. These inaccuracies and low motion field density lead to poor boundary localization of segmented objects in the compressed domain. In order to obtain a high density motion field, we propose an MV integration scheme that can operate in a coherent or non-coherent fashion and on a block or pixel basis.

3.2.1.1. Coherent vs. noncoherent MV Integration

Due to the variable block size of encoded MVs, we first normalize (by splitting a block and assigning MV of the parent block to all child blocks) all blocks and their MVs to the minimum block size supported by a video codec, which we denote by Z (e.g., $Z = 1$ for pixel-based MV, $Z = 4$ in H.264). Let $\mathbf{MV}_{x,y}(t) = \left(MV_{x,y}^{X}(t), MV_{x,y}^{Y}(t)\right)$ denote the MV for a block in frame t, and (x, y) denote the center coordinates of this block divided by Z. As in [20], an integrated MV field can be obtained by summing up MVs over several previous frames:

$$\mathbf{MV}_{x,y}^{COH}(t) = \sum_{i=0}^{N-1} \mathbf{MV}_{x_i,y_i}(t-i) \qquad \text{Equation 3.7}$$

$$(x_{i+1}, y_{i+1}) = (x_i, y_i) + \textit{round}\left(\frac{1}{Z} \cdot \left(MV_{x_i,y_i}^{X}(t-i), MV_{x_i,y_i}^{Y}(t-i)\right)\right) \qquad \text{Equation 3.8}$$

where N is the number of frames over which integration is performed, and (x_i, y_i) represent the block's center coordinates in frame $(t-i)$. We refer to this process as *coherent motion integration*.

The problem with this type of MV integration is that repetitive motion (e.g., bouncing ping-pong ball in *Table Tennis*) leads to motion cancellation, which may cause a moving region to become undetectable in certain frames. Hence, we introduce another way to integrate MVs, which we call *noncoherent motion integration*. This is done by adding up the MV magnitude over successive coherent integrations:

$$\mathbf{MV}_{x,y}^{NCO}(t) = \mathbf{MV}_{x,y}(t) \cdot \left(1 + \sum_{i=1}^{M} \frac{\left\|\mathbf{MV}_{x_i,y_i}(t-i)\right\|}{\left\|\mathbf{MV}_{x,y}(t)\right\|}\right) \qquad \text{Equation 3.9}$$

where M is the total number of frames used in this noncoherent integration, and (x_i, y_i) is computed by Eq. 3.8. This way, when motion phase cancels out the motion of a particular region over a certain time period, motion magnitude should still help detect and segment that region.

To harvest the benefits of both types of MV integration, we also propose a combination of coherent and noncoherent integration as shown in Fig. 3.7, where coherent integration is applied over blocks of N frames, while noncoherent integration is applied over M groups of N frames (i.e., $N \times M$ frames in total). We refer to such a scheme as (N, M) combined MV integration.

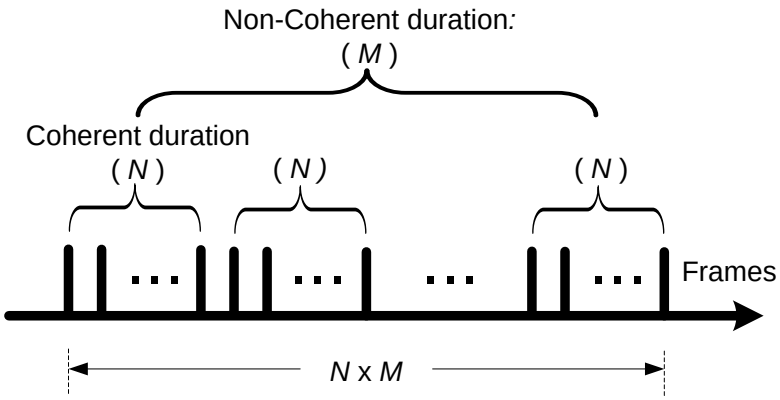


Figure 3.7: Combined coherent and non-coherent MV integration.

3.2.1.2. Block- vs. Pixel-based MV Integration

MV integration can be block-based or pixel-based. Block-based integration is faster and requires less memory, which are its two biggest advantages. It is also less accurate than pixel-based integration, and leads to low resolution region boundaries, which has been an issue for all previous compressed-domain segmentation methods [18][20][27][78][80-81]. Block-based MV integration can be explained with the help of Fig. 3.8. Here, (x, y) is the center of block B_4 in frame t. The reference block in the previous frame overlaps four blocks (B_1 - B_4). One way to achieve block-based MV integration is to add the weighted average of MVs of blocks B_1 through B_4 from the reference frame $t-1$ to the MV of B_4 in frame t, as in [20]. Another way, used in our system, is to add the MV of B_4 in frame t to the MV of the block in frame $t-1$ with which its reference block has the largest overlap. This is also known as Forward Dominant Vector Selection (FDVS) [125]. In Fig. 3.8, this is B_3 with center (x', y'). This operation is easily executed via rounding in Eq. 3.8.

The rounding operation adds quantization noise to the MVs, so we cannot expect high accuracy from block-based MV integration. But in our system, block-based integration is used only for coarse segmentation, to get a rough idea where moving regions are and to identify boundary regions between them. For higher accuracy needed in boundary refinement we use pixel-based MV integration, which can also be illustrated

in Fig. 3.8. Now, each pixel in B_4 in frame t is assigned the MV of that block and to each of these MVs we add the MV of the reference pixel in frame $t-1$. This means that the pixels from the upper-left corner of B_4 in frame t will acquire the MV of B_1 in frame $t-1$, the pixels in the upper right corner of B_4 in frame $t$will acquire the MV of B_2 in frame $t-1$ and so on. For this kind of integration sub-pixel accurate MVs are rounded to pixel accuracy, which again adds some quantization noise to the MVs, but still allows for boundary region segmentation with pixel precision.

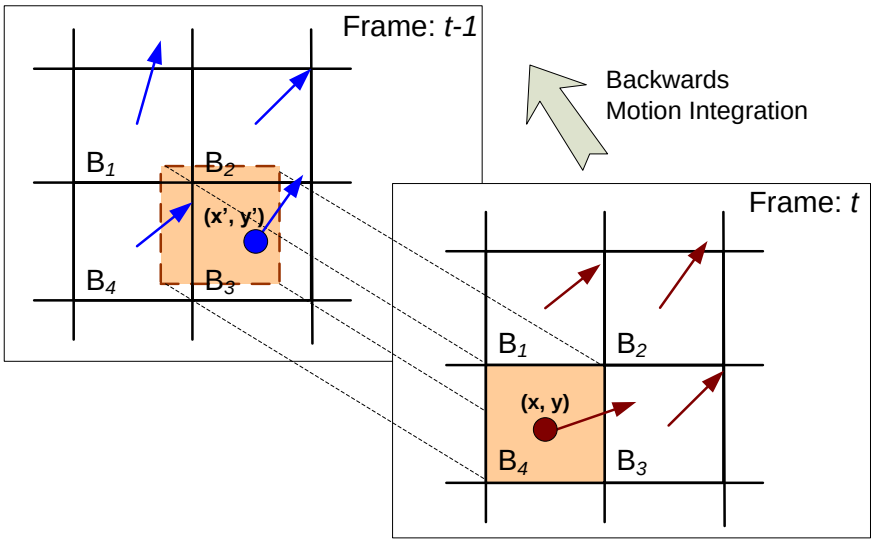


Figure 3.8: Block- and pixel-based MV integration.

3.2.2. Segmentation

To balance the computational complexity and performance, we adopt a coarse-to-fine strategy for segmenting moving regions in the compressed domain. In the method described in this section, all processing is done in the compressed domain and using only motion information. The segmentation process consists of four major steps: 1) Coarsely segmenting moving regions on a block basis; 2) Identifying boundary regions; 3) Increasing motion density within boundary regions; and 4) Refining region boundaries using contour evolution.

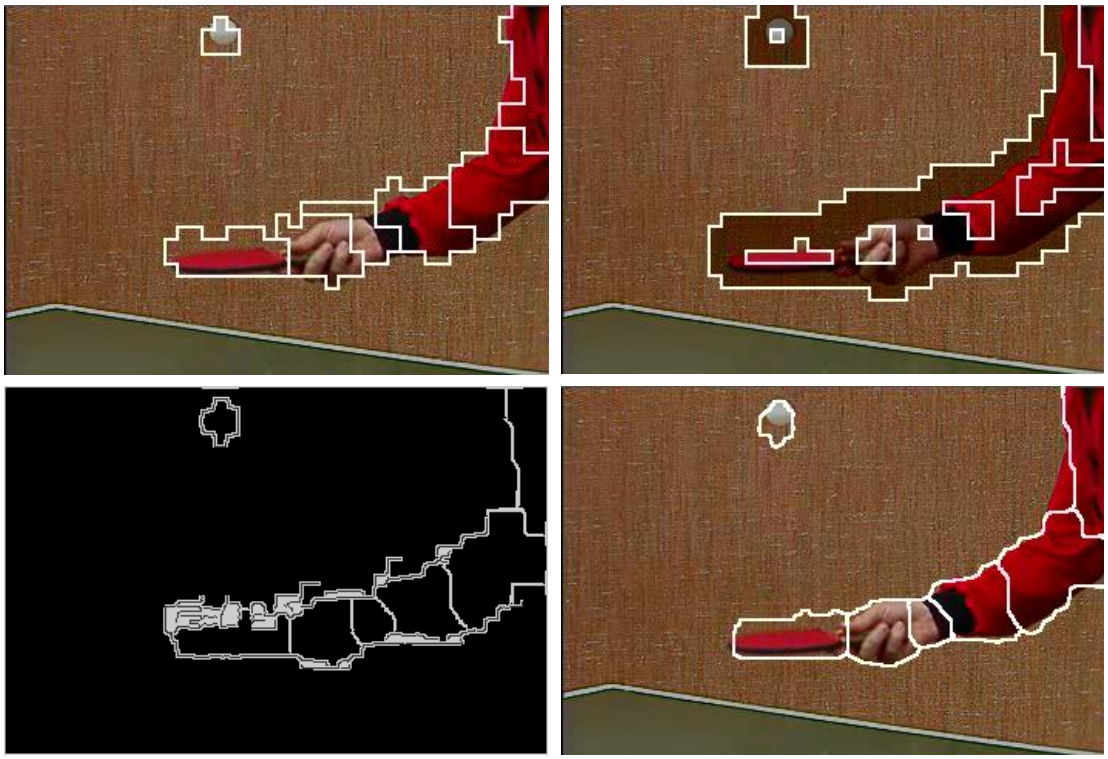


Figure 3.9: Step-by-step segmentation results. [Top Left]: coarse segmentation, [Top Right]: boundary regions (darker areas), [Bottom Left]: contour evolution, [Bottom Right]: boundary refinement.

Figure 3.9 illustrates the step-by-step results of moving region segmentation in frame #18 of *Table Tennis*, where $(N, M) = (2, 2)$ combined block-based MV integration is used for coarse segmentation, and $(N, M) = (3, 3)$ combined pixel-based MV integration is used for boundary refinement. Decoded MVs associated with each block are normalized according to the temporal distance and direction indicated by the reference frame index, and mapped to the minimum block size supported by the codec — 8×8 blocks in our case, since we are using MPEG-4. We integrate these normalized MVs on a block basis and detect moving regions by using the coarse segmentation algorithm from Section 3.1, which combines iterative *k*-means clustering and a motion consistency model (top left of Fig. 3.9).

We further identify boundary regions between moving regions based on the result of coarse segmentation. A block is flagged as a *boundary block* if it has 8-adjacency neighbours from more than one region. These boundary blocks form *boundary regions* where real moving region boundaries are likely to be found (top right of Fig. 3.9). Boundary regions are subject to boundary refinement. The remaining regions are classified as *interior regions*.

In order to find real region boundaries with pixel precision, we first use pixel-based MV integration to increase MV density in boundary regions. Next, a Canny edge detector on motion magnitude is used to identify edges inside boundary regions. Then, an interior region contour is evolved via morphological erosion [82] of the boundary region using a 3×3 structuring element (bottom left of Fig. 3.9). In this process some contours of neighbouring regions may meet, in which case the pixel-wise boundary between these regions is identified. In other cases contours do not meet due to MV noise in the boundary region, so we use a morphological closing followed by opening to get smooth contours (bottom right of Fig. 3.9).

3.2.3. Evaluation

The performance of the proposed segmentation algorithm was tested in an MPEG-4 compressed domain. Several test sequences (*Flower Garden*, *Table Tennis*, *Hall Monitor*, *Ice* and *Coastguard*) at 30 fps, CIF (352×288) and SIF (352×240) resolutions were compressed at 512 kbps using the Xvid MPEG-4 video coder, with an IPPP GOP structure. All tests were done on a fairly standard desktop PC with an Intel Pentium D processor at 3GHz and 2GB RAM.

We first illustrate the benefits of MV integration. Figure 3.10 shows how segmentation sensitivity can benefit from block-based MV integration, where sample frames are from *Table Tennis*. The top left of Fig. 3.10 shows Frame #9 reconstructed by the Xvid MPEG-4 decoder and the top right of Fig. 3.10 shows its corresponding normalized motion vector field without integration, where the brightness represents MV magnitude. We can observe some very vague moving regions. The bottom left of Fig. 3.10 shows the motion field after coherently integrating MVs over 4 previous frames (from frame #6 to #9). The regions are more visible compared to the previous motion

field without integration. The motion field is further sharpened after coherent MV integration over 8 frames (from frame #2 to #9), as shown in the bottom right of Fig. 3.10. The moving regions appear much more distinct from the static background and can be easily segmented. The only remaining issue is the boundary precision. Since all MV integrations in this example are performed on a block basis, moving regions end up with coarse (block-based) boundaries.

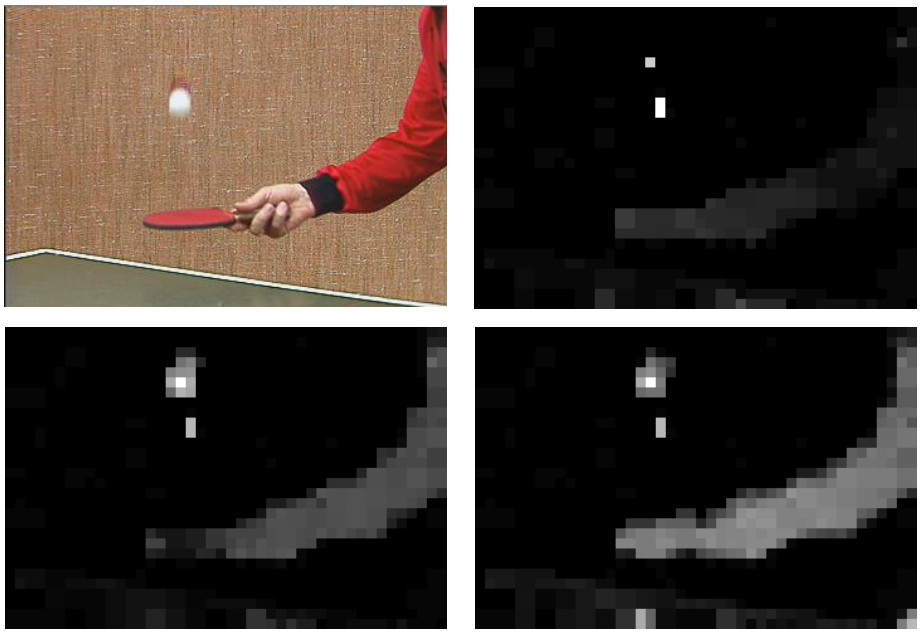
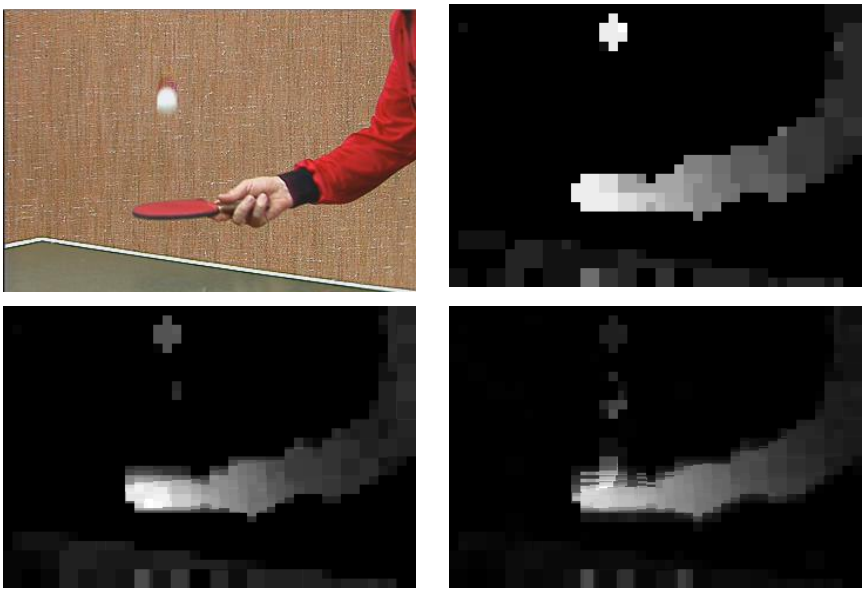
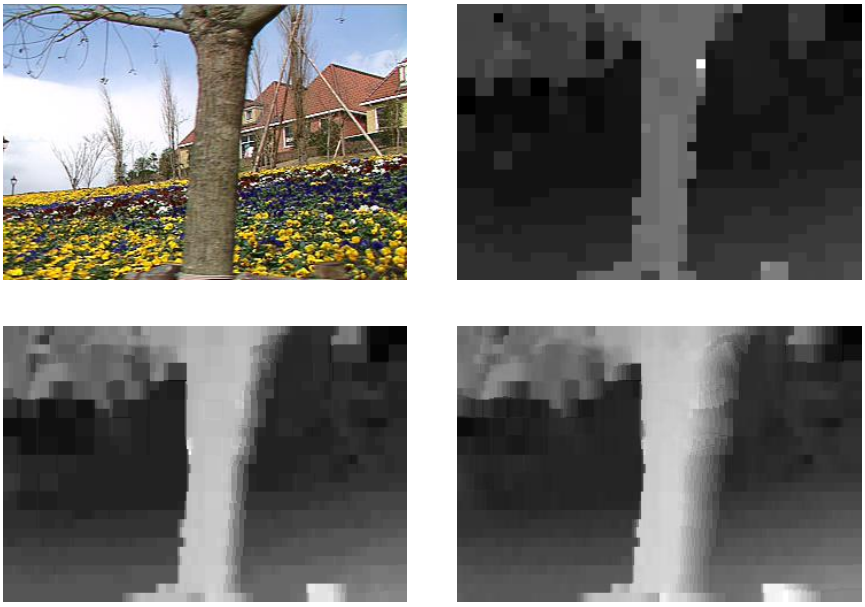


Figure 3.10: Increasing segmentation sensitivity by block-based MV integration on *Table Tennis*, [Top Left]: reconstructed frame #9, [Top Right]: motion field without integration, [Bottom Left]: motion field with $(N, M) = (1, 4)$ MV integration, [Bottom Right]: motion field with $(N, M) = (1, 8)$ MV integration.



(a)



(b)

Figure 3.11: Increasing motion field density by pixel-based MV integration on (a) *Table Tennis* and (b) *Flower Garden*. [Top Left]: reconstructed frame #9, [Top Right]: motion field without integration, [Bottom Left]: motion field with $(N, M) = (1, 4)$ MV integration, [Bottom Right]: motion field with $(N, M) = (1, 8)$ MV integration.

Figure 3.11 illustrates how the density of the motion field can be increased by pixel-based MV integration on *Flower Garden*. The top left of Fig. 3.11 shows Frame #9 reconstructed by the Xvid MPEG-4 decoder and the top right of Fig. 3.11 shows the motion vector field, where the brightness is computed based on the magnitude of the corresponding MV. Since no MV integration is involved, we can observe a very sparse motion field. The bottom left of Fig. 3.11 shows a denser motion field after coherently integrating MVs over 4 frames (from frame #6 to #9). We can vaguely figure out the tree trunk and there is a significant reduction in block-like edges. After coherent MV integration over 8 frames, as shown in the bottom right of Fig. 3.11 (from frame #2 to #9), the motion field appears much smoother, especially on the front line of the moving object (left boundary of the tree in this example), the region boundary is quite distinctive compared to its neighbouring regions. We can use common edge detectors (e.g. Canny) on motion magnitude to identify these boundaries and get a step closer to segmentation with pixel precision.

Figure 3.12: Ground truth and region masks – *Flower Garden* and *Table Tennis*. [Left]: original frame, [Middle]: manually segmented regions, [Right]: region masks.

We further quantitatively evaluate the proposed region segmentation on *Flower Garden* and *Table Tennis*. We manually labeled all the pixels of the fastest moving

regions in the first 20 frames (as shown in Fig. 3.12, tree in *Flower Garden*, player's hand and ball in *Table Tennis*) as the moving object ground truth and tested how accurately these regions can be segmented.

By counting the pixels correctly identified as moving region pixels (True Positives – TP), the pixels correctly identified as the background (True Negatives – TN), the pixels wrongly identified as moving region pixels (False Positives – FP), and the pixels wrongly identified as background (False Negatives – FN), Sensitivity, Specificity and False Alarm Rate (FAR) can be computed as Sensitivity = TP / (TP + FN), Specificity = TN / (TN + FP) and FAR = FN / (TP + FN).

Using different MV integration configurations we constructed three segmentation methods with different complexities, and compared their performance.

Method-1) (Low complexity): the simplest segmentation method, where only coarse segmentation is used without MV integration.

Method-2) (Medium complexity): $(N, M) = (1,2)$ block-based MV integration for coarse segmentation, followed by $(N, M) = (1,3)$ pixel-based MV integration for boundary refinement.

Method-3) (High complexity): $(N, M) = (2,2)$ block-based MV integration for coarse segmentation, and $(N, M) = (3,3)$ pixel-based MV integration for boundary refinement.

Table 3.1 shows the average sensitivity, specificity and FAR of segmentation for the three methods. We can see that both segmentation sensitivity and specificity benefit from MV integration over multiple frames and **Method-3** has a FAR twice as low as **Method-1** on both sequences.

For further comparisons, we plot sensitivity and specificity values of three segmentation methods for the first 20 frames of *Flower Garden* and *Table Tennis* shown in Fig. 3.13. The overall performance of **Method-3** is generally the best. Please note that in *Table Tennis*, where MVs appear more inconsistent over frames than *Flower Garden*, the sensitivity curve of **Method-3** appears much smoother than that of **Method-1** and **Method-2.**

Table 3.1: Sensitivity, specificity and false alarm rate

Sequence		*Flower Garden*	*Table Tennis*
Sensitivity (%)	**Method-1**	97.08	77.69
	Method-2	98.36	86.23
	Method-3	98.86	89.95
Specificity (%)	**Method-1**	92.95	95.57
	Method-2	93.58	96.15
	Method-3	94.82	96.29
FAR (%)	**Method-1**	2.92	22.31
	Method-2	1.74	13.77
	Method-3	1.14	10.05

Table 3.2: Average segmentation time per frame

Sequence	**Time in milliseconds**			
	Xvid Decoding	**Method-1**	**Method-2**	**Method-3**
Table Tennis	6.19	7.12	39.08	47.74
Coastguard	6.68	6.35	36.36	51.23
Flower Garden	5.98	6.17	42.86	59.99
Ice	7.01	7.34	45.95	55.57
Hall Monitor	7.22	6.99	37.58	52.06

Finally, Fig. 3.14 shows region segmentation results of the five test sequences for three methods, which illustrate the improvement on segmentation sensitivity and region boundaries using MV integration. Images from left to right are the segmentation results from **Method-1**, **Method-2**, and **Method-3**, respectively. We can observe the coarseness of region segmentation by **Method-1**, while region boundaries are refined and much closer to real object boundaries by pixel-based MV integration in **Method-3.**

We also investigated the complexity of the proposed algorithms by measuring their average execution time within the setup described above. The results for the three methods with different integration configurations are reported in Table 3.2. We also

report the Xvid MPEG-4 decoding time per frame, which is the amount of time we can save using the proposed compressed domain approach compared to other approaches based on pixel-domain processing (which involves decoding). Note that while Method-3 is the most accurate among the three methods, its processing requires 40~50 milliseconds per frame. In order to achieve 30 frames per second, the algorithm would need further optimization, possibly on an advanced platform such as GPU.

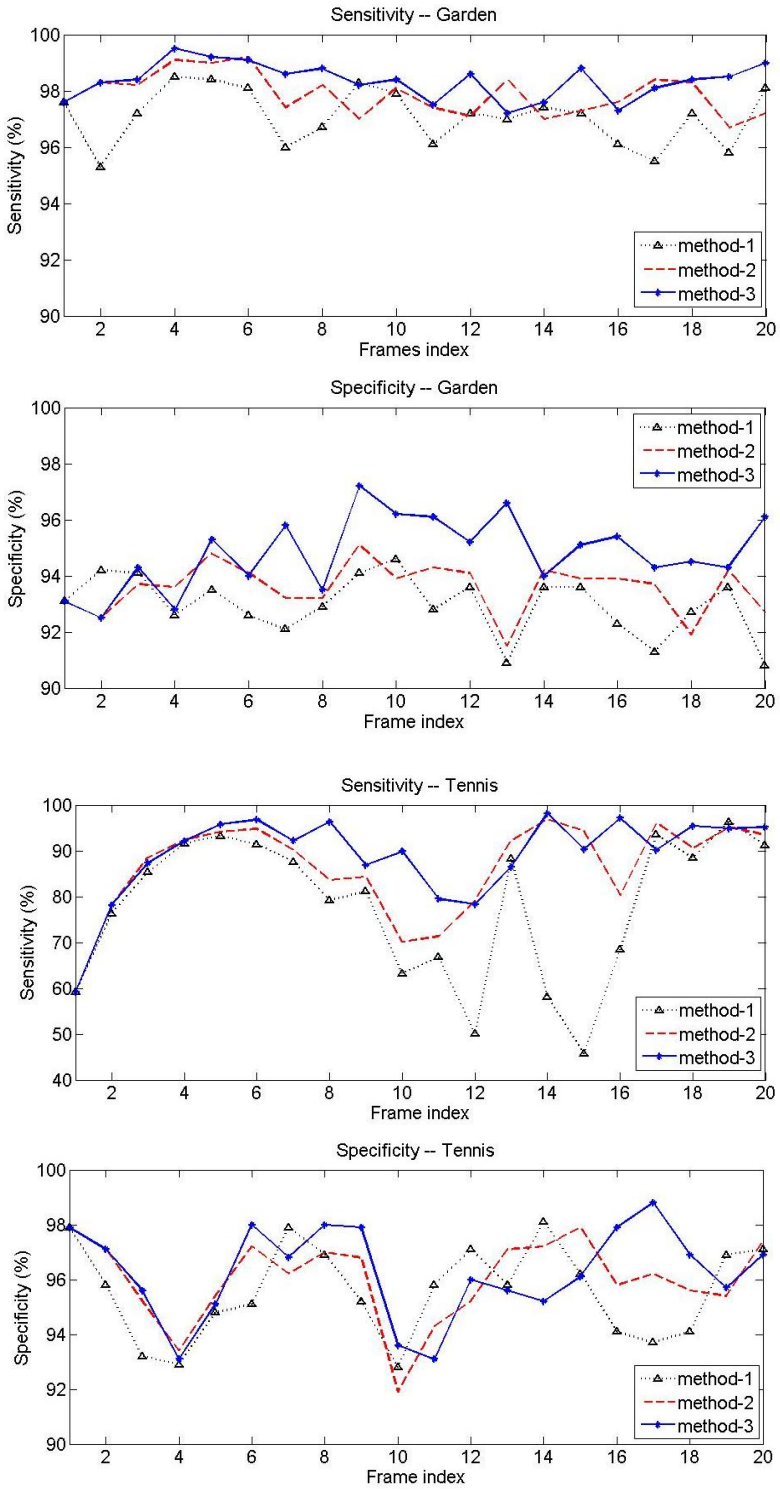


Figure 3.13: Sensitivity and specificity of region segmentation – *Flower Garden* and *Table Tennis*.

Figure 3.14: Region segmentation results of five sequences for the three methods. [Left to right], Method-1, Method-2, and Method-3, respectively. [Top row to bottom row]: *Table Tennis*, *Coastguard*, *Flower Garden, Ice,* and *Hall Monitor,* respectively.

3.3. Coarse-to-Fine Moving Region Segmentation

In this section, we address object boundary localization problem by proposing a combined compressed-domain and pixel-domain segmentation framework. The block-based *k*-means segmentation algorithm described in Section 3.1 enables us to coarsely isolate moving regions at block precision. Subsequently, we identify the coarse boundaries between moving regions and then perform fine segmentation of these boundaries using the local colour and edge information. This way, we direct the computational effort associated with pixel-based processing where it is most needed — near the boundaries of moving regions. This leads to a *coarse-to-fine* segmentation algorithm, as we present below. This framework is also presented in our previous work [27].

The coarse-to-fine segmentation algorithm consists of three phases:

1. Block-based coarse segmentation from motion, described in Section 3.1.
2. Block-based boundary region identification.
3. Pixel-based fine segmentation of boundary regions using colour and edge information.

In the first phase, described in the Section 3.1, we make use of the block-based motion vectors (MVs) available in the compressed bitstream to isolate different moving regions. At the end of this phase a coarse segmentation map is generated from which we extract the boundaries of the moving regions in the second phase. In the third phase, a fine pixel-based segmentation is applied to the block-based boundaries identified in the second phase to give a more accurate segmentation map of the moving regions. Therefore, computationally demanding pixel-based processing is only performed in the last phase of the algorithm.

Since the first two phases rely on the existing MVs, they can be applied to P- or B-frames, but not to I-frames. While it is possible to use a purely spatial segmentation method for the I-frames, there is really no need for it. The number of I-frames is generally small compared to P- or B-frames, and they are usually far apart, so any object

of interest that appears in an I-frame would also appear in one or more intermediate P- or B-frames.

3.3.1. Boundary Region Refinement

An important step prior to fine region segmentation is to classify blocks based on the result of coarse segmentation. Within each region, each block is classified in terms of its position as either an *interior block* or a *boundary block*. A block is classified as a *boundary block* if it has 8-adjacency neighbours from more than one region. Otherwise, it is classified as an *interior block*.

Interior blocks are not subject to fine segmentation. Instead, they provide local colour statistics that are used in subsequent region growing. A region consisting of interior blocks is called an *interior region*, while a region made up of boundary blocks is called a *boundary region*. These boundary regions will be re-segmented based on local edge and colour information.

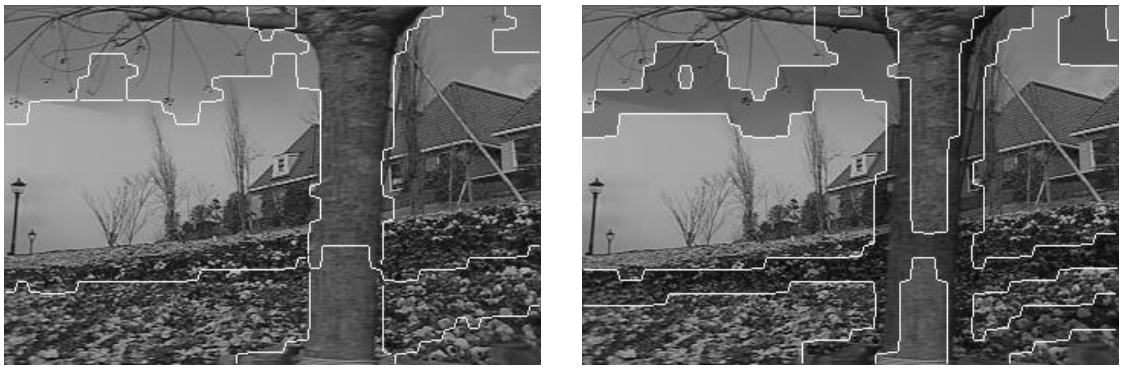


Figure 3.15: Boundaries of the coarse segmentation map (left) and boundary regions (right).

Figure 3.15 shows an example of boundary regions for frame #2 of *Flower Garden*. The left part of the figure shows the boundaries of the coarse segmentation map from Fig. 3.5 overlaid on top of the actual frame, while the right part shows the boundary blocks identified from the coarse segmentation map. These blocks are shown with a slightly darker grey level. They occupy only 22% of the frame in this example,

which means that the computationally intensive pixel-based segmentation will only be needed in a relatively small part of the frame.

3.3.2. Colour Clustering Initialization in the I-Frame

As stated above, I-frames are not segmented in the proposed method. Instead, they are used to help segment other frames by providing initial colour clusters that represent the dominant colours of a particular scene. The refinement of colour clusters is done in subsequent P- or B-frames via *k*-means clustering.

The initialization of colour clusters starts by quantizing the reconstructed I-frame into YUV colour bins of size 8×8×8. Then we create a histogram that counts how many pixels are included in each bin. Using this histogram we can reduce the number of bins by choosing a subset of bins that account for a certain percentage of the pixels in the frame. This way, we can choose the number of bins and their locations in the YUV space that are appropriate for a particular scene. The colour bins obtained in this way are taken as the initial set of clusters for subsequent refinement via *k*-means clustering in the following P- or B-frames.

3.3.3. Fine Segmentation in P- or B- Frames

The final segmentation phase starts with edge detection in boundary regions using the *Canny* edge detector. After this, a region growing process is initiated where the local edge and colour information are used to merge pixels in the boundary region into one of the neighbouring interior regions.

A sliding window of size 3×3 blocks (24×24 pixels when using 8×8 pixel blocks in MPEG-4 video) is used to calculate local colour statistics. This window is moved in the vicinity of boundary regions and region growing happens when the center block of the sliding window is located on an interior block which borders a boundary region in its 4-adjacency neighbours. An example can be seen in Fig. 3.16. Region growing consists of the following steps:

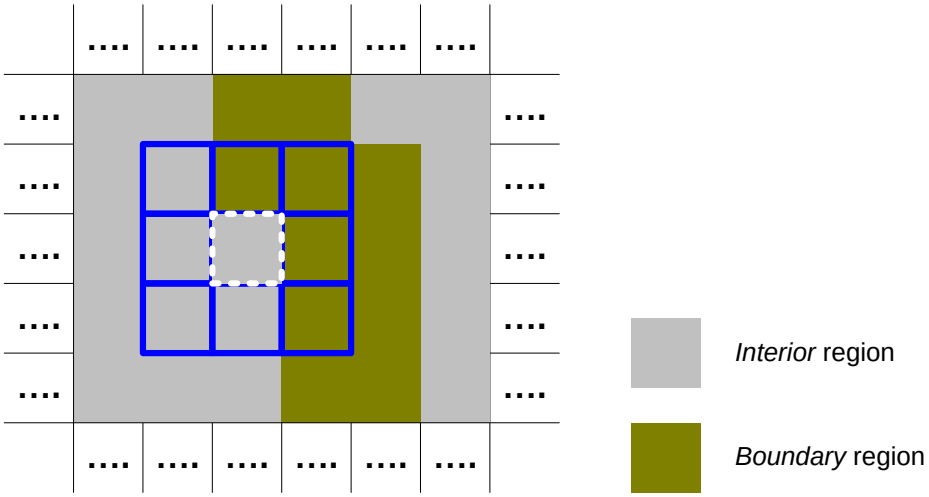


Figure 3.16: Window location for region growing.

Step 1. Calculate the region growing threshold TH for the interior region within the window. TH is defined as the weighted Root Mean-Square (RMS) deviation [51] of interior region pixels values (y_n, u_n, v_n) within the window, from their mean values $(\bar{y}, \bar{u}, \bar{v})$:

$$TH = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(w_y(y_n - \bar{y})^2 + w_u(u_n - \bar{u})^2 + w_v(v_n - \bar{v})^2\right)} \qquad \text{Equation 3.10}$$

Here, N is the number of interior pixels within the window, while the weights w_y, w_u, and w_v are chosen to balance the influence of color components on the color distance. In our experiments, $(w_y, w_u, w_v) = (1,2,2)$.

Step 2. Group the boundary region pixel (y_n, u_n, v_n) within the window into the neighbouring interior region if the pixel is not on the edge, and the color distance between the pixel and mean of the interior region is less than TH:

$$\sqrt{w_y(y_n - \bar{y})^2 + w_u(u_n - \bar{u})^2 + w_v(v_n - \bar{v})^2} \leq TH \qquad \text{Equation 3.11}$$

Step 3. Repeat steps 1) and 2) until no more pixels can be grouped based on local colour statistics. Then grow all existing regions up to the edges.

Step 4. Group all remaining pixels in boundary regions, including edge pixels, to the neighbouring interior region with the closest mean in the sense of the colour distance in Eq. 3.10.

Figure 3.17 illustrates the step-by-step results of fine segmentation for frame #2 of *Flower Garden*.

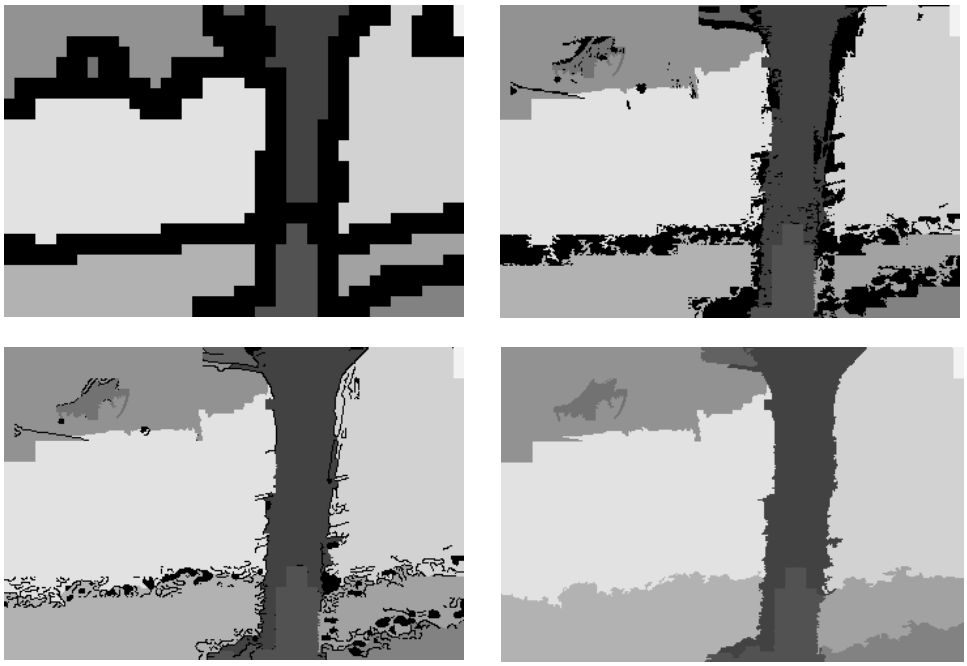


Figure 3.17: Region growing results after step 1 (top left), step 2 (top right), step 3 (bottom left) and step 4 (bottom right).

3.3.4. Evaluation

The proposed coarse-to-fine moving region segmentation has been implemented in the XviD MPEG-4 decoder, and tested on three standard YUV SIF (352×240) sequences at 30 fps: *Flower Garden*, *Table Tennis*, and *Football*.

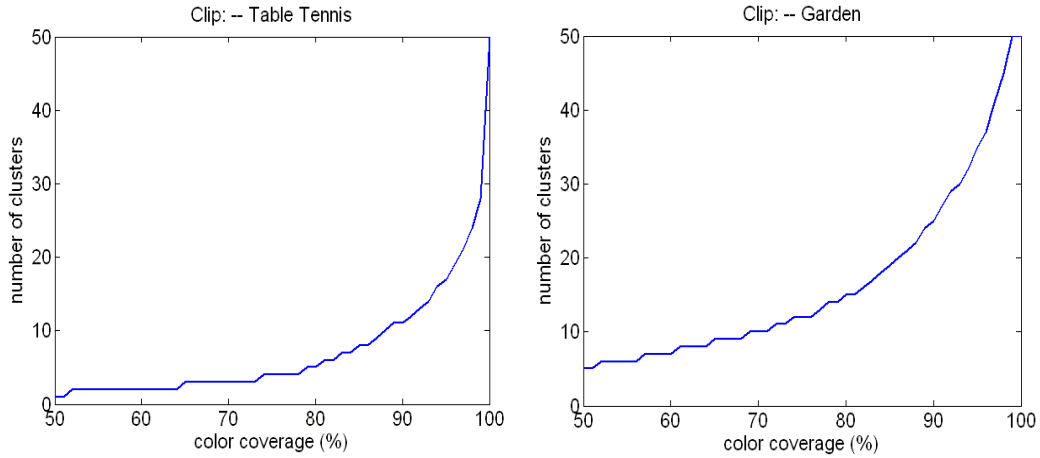


Figure 3.18: Number of clusters vs. color coverage

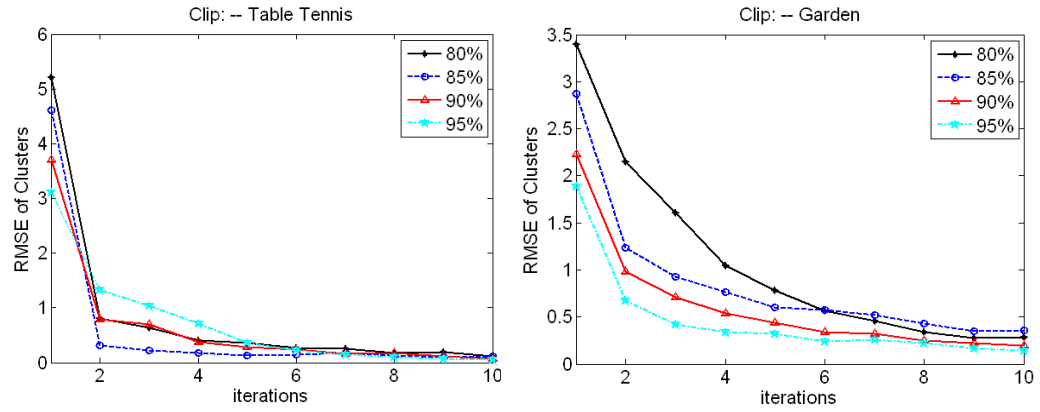


Figure 3.19: Number of *k*-means iterations vs. cluster RMSE, for coverage of 80%, 85%, 90%, and 95%.

The number of colour clusters that is appropriate for a particular sequence depends on the characteristics of that sequence. We use two sequences, *Table Tennis* and *Flower Garden*, to show the relationship between the number of clusters and the percentage of pixels that fall into those clusters (called “colour coverage”) as shown in Fig. 3.18.

In our experiments, we used the number of clusters that gave the color coverage of 90%, which turned out to give a good compromise between computational complexity (which increases with the number of clusters) and segmentation accuracy. From Fig. 3.18, to achieve 90% coverage, 11 clusters are needed for *Table Tennis* and 25 clusters for *Flower Garden*. This is not surprising, because *Flower Garden* has a larger variety of colors. Choosing the color clusters based on this simple coverage criterion allows our method to pick the number of clusters that is appropriate for each sequence.

Next, Fig. 3.19 displays the relationship between the number of *k*-means iterations and the average Root Mean-Square Error (RMSE) of color clusters for *Table Tennis* and *Flower Garden*. With 90% coverage, five iterations are enough to bring RMSE below 1, which is low enough for our purposes. These five *k*-means iterations are carried out in the I-frame, while only one iteration is performed in each subsequent P- or B-frame to update the clusters.

Finally, Fig. 3.20 shows the visual comparison between coarse and fine segmentation for the frames from Fig. 3.5. It can be observed that region boundaries obtained by fine segmentation are very close to the real boundaries.

In addition to visual comparison, we provide a quantitative comparison between coarse and fine segmentation in Tables 3.3 and 3.4 for sequences compressed at 512 kbps and 256 kbps, respectively. To get the ground truth, we manually labelled all the pixels of moving regions of interest – tree in *Flower Garden*, players in *Football*, player and ball in *Table Tennis*. Based on the ground truth, we counted the pixels in the moving regions that are correctly segmented as moving regions (these pixels are called True Positives – TP), the pixels from the “stationary” region (the region with the lowest motion, which can be interpreted as background) that are wrongly identified as moving region pixels (these are called False Positives – FP), stationary region pixels correctly identified as stationary region pixels (these are True Negatives – TN), and moving region pixels wrongly identified as stationary pixels (these are False Negatives – FN). Using these counts, we computed *Sensitivity*, *Specificity* and *False Alarm Rate* (*FAR*) as explained in Section 3.2.3.

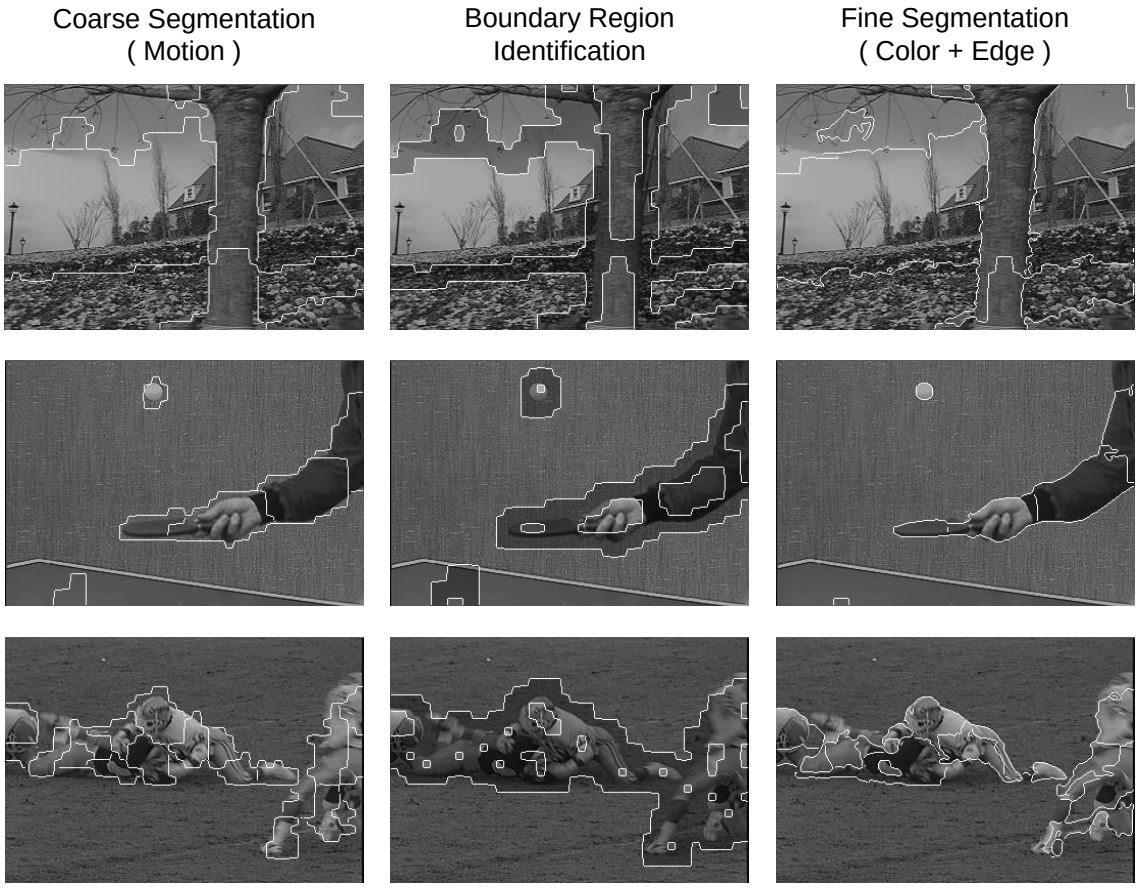


Figure 3.20: Coarse-to-fine segmentation [Top to Bottom]: *Flower Garden* (frame #2), *Table Tennis* (frame #4), and *Football* (frame #2).

Table 3.3: Coarse (C) vs. Fine (F) segmentation at 512 kbps

Sequence	*Sens.* (%)		*Spec.* (%)		*FAR* (%)	
	C	F	C	F	C	F
Flower Garden	91.6	94.9	94.5	98.1	8.4	5.1
Table Tennis	80.2	93.8	97.1	99.1	19.8	6.2
Football	86.1	90.3	92.0	96.1	13.9	9.7

Table 3.4: Coarse (C) vs. Fine (F) segmentation at 256 kbps

Sequence	*Sens.* (%)		*Spec.* (%)		*FAR* (%)	
	C	F	C	F	C	F
Flower Garden	88.6	92.5	92.3	97.1	11.4	7.5
Table Tennis	79.9	91.5	96.1	98.6	20.1	8.5
Football	84.8	88.9	90.8	95.3	15.2	11.1

As seen in the tables, coarse-to-fine region segmentation can improve sensitivity up to 13%, while reducing FAR 2-3 times compared to the coarse segmentation. Note that the segmentation accuracy reduces at lower bit rates, as expected, since the motion vectors and pixel values are less accurate at lower bit rates.

Finally, we assess the speed of the proposed coarse-to-fine segmentation method by measuring its execution time on a fairly standard desktop PC with an Intel Pentium CPU at 3.0 GHz and 2 GB of RAM. On average, it takes 5.3ms to initialize the colour clusters in the I-Frame, 4.0ms to perform coarse segmentation, and 4.1ms to perform subsequent fine segmentation.

3.4. Coarse-to-Fine Segmentation Using Markov Random Fields

The coarse-to-fine segmentation approach presented in Section 3.3 effectively balances complexity and accuracy. However, maintaining a consistent number of segmented regions across frames is still a challenge. Under- and over-segmentation are two common issues in this regard. Similar problem exists in other joint compressed-domain and pixel-domain approaches [27][51][71].

In this section, we address the segmentation consistency issue by employing a Markov Random Field (MRF) motion model within a coarse-to-fine segmentation framework. Some of the material presented in this section can be found in our earlier work [73] [75]. A distinctive feature of our method is the use of MV quantization based on local motion similarity to find the most likely number of moving objects/regions and use the statistics of the resulting clusters to initialize prior probabilities for subsequent

Markov Random Field (MRF) classification. This way, the proposed method is able to overcome some of the difficulties faced by previous methods, such as over-segmentation [14-15], under-segmentation [67], and segmented region inconsistency [27][51[71]. Further, maintaining coarse-to-fine framework yields a segmentation map with more accurate region boundaries than can be achieved by purely compressed domain methods [67-68], while still having a much lower complexity than pixel domain methods [61][69-70].

A block diagram of the proposed segmentation system is shown in Fig. 3.21. The system incorporates two major segmentation components: coarse segmentation from motion, which is carried out in the compressed domain, and fine segmentation, performed in the pixel domain near moving region boundaries. GME and GMC, along with MV quantization, serve as the foundations for coarse segmentation, and facilitate the computation of the priors for Markov Random Field (MRF) MV classification.

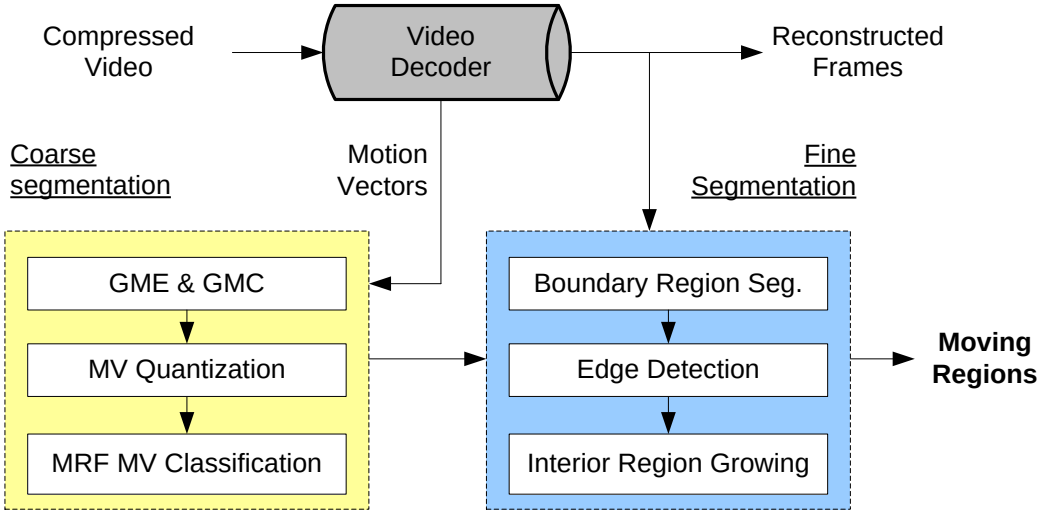


Figure 3.21: Overview of the MRF moving region segmentation system.

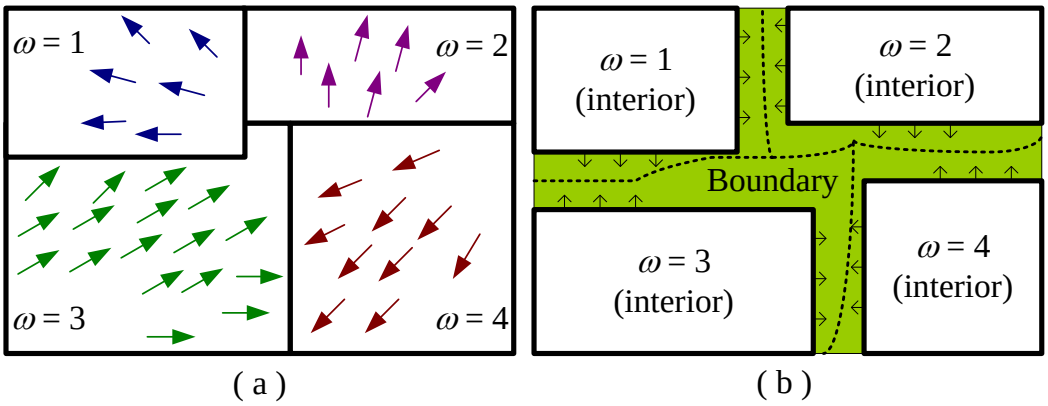


Figure 3.22: (a): Illustration of the MRF motion model, (b): interior regions grow towards boundaries.

3.4.1. Markov Random Field Motion Model

The approach to coarse motion segmentation is based on a Markov Random Field (MRF) motion model [15], [70], [74]. As illustrated in Fig. 22(a), in a MRF motion model, motion vectors $\mathbf{MV} = (MV^X, MV^Y)$ within a given moving region ω follow a conditional distribution $P(\mathbf{MV} \mid \omega)$, while region labels ($\omega$'s) follow a 2D MRF distribution based on a given neighbourhood system. The goal is to infer region labels (ω's) from the observed MV field.

To simplify calculations, we assume that within each region, MVs form an independent bivariate Gaussian process. Under this assumption, the likelihood function for the j-th block in the frame is.

$$P(\mathbf{MV}_j|\omega_j) = \frac{1}{\sqrt{2\pi}\,\sigma_{\omega_j}^X\,\sigma_{\omega_j}^Y} \cdot \exp\left(-\frac{1}{2}\left(\frac{\left(MV_{\omega_j}^X - \mu_{\omega_j}^X\right)^2}{\left(\sigma_{\omega_j}^X\right)^2} + \frac{\left(MV_{\omega_j}^Y - \mu_{\omega_j}^Y\right)^2}{\left(\sigma_{\omega_j}^Y\right)^2}\right)\right) \quad \text{Equation 3.12}$$

where $\mu_{\omega_j}^X$ and $\mu_{\omega_j}^Y$ are the means of the horizontal and vertical MV components within the region labeled ω_j, while $\sigma_{\omega_j}^X$ and $\sigma_{\omega_j}^Y$ are the corresponding standard deviations. The dependence among the labels of neighbouring blocks is modeled by a MRF, which follows the Gibbs distribution:

$$P(\omega_j) = \frac{1}{Z}\prod_C exp(-V(C)) \qquad \text{Equation 3.13}$$

where Z is the normalizing constant ensuring that $\sum P(\omega_j) = 1$, C is a *clique* (a set of neighbouring blocks) and $V(C)$ is the *clique potential*. We only consider the 4-adjacency cliques. In other words, two blocks form a clique if one is immediately to the North, South, East or West of the other, as illustrated in Fig. 3.23.

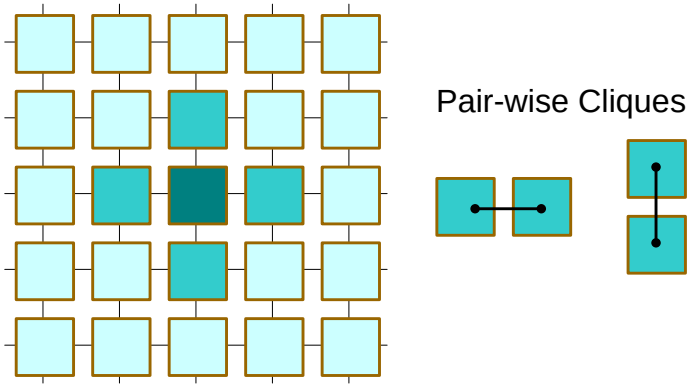


Figure 3.23: First-order MRF system and clique configuration.

If ω_1 and ω_2 are the region labels of the two blocks in clique C, the potential of C is defined to be

$$V(C) = \begin{cases} -\beta, & \text{if } \omega_1 = \omega_2, \\ +\beta, & \text{otherwise,} \end{cases} \qquad \text{Equation 3.14}$$

where $\beta > 0$ is a parameter controlling the homogeneity of the regions. Based on Eq. 3.14 and 3.15, nearest neighbours are more likely to have the same region label.

3.4.2. MV Quantization and MRF Parameter Estimation

The presence of camera motion often worsens the performance of these segmentation approaches, and objects may be over-segmented due to motion bias introduced by camera movement. Prior to MV quantization, GME and GMC are employed to remove the influence of camera motion on the MV field.

We use a perspective model with eight parameters, $\mathbf{m} = [m_0, \cdots, m_7]$, to represent global motion. This model describes the 2D projection of the 3D motion of a planar surface, so it is often used to model the motion of the background, which is assumed far from the camera. Estimating global motion parameters ($\mathbf{m}$) from noisy MV fields involves two steps, often performed iteratively: outlier removal and parameter estimation. In this section, we use LSS-ME [11] for estimating $\mathbf{m}$ due to its superior performance when the 8-parameter perspective motion model is used [26].

Let $\mathbf{m}_t$ be the vector of estimated GM parameters from LSS-ME in frame t. The global motion can be compensated from the MV at location (x, y) in frame t by:

$$\mathbf{MV}_{\boldsymbol{x,y}}^{res}(t) = \mathbf{MV}_{x,y}(t) - \mathbf{MV}_{x,y}(t; \mathbf{m}_t) \qquad \text{Equation 3.15}$$

where $\mathbf{MV}_{\boldsymbol{x,y}}^{res}(t)$ is the compensated MV at location (x, y), and $\mathbf{MV}_{x,y}(t; \mathbf{m}_t)$ is the MV generated by the global motion model $\mathbf{m}_t$. Motion quantization is then conducted on $\mathbf{MV}_{\boldsymbol{x,y}}^{res}(t)$.

The main difficulty in MRF segmentation is to determine the parameters that specify the MRF, particularly the number of motion segments and their statistics. Our approach is to first perform vector quantization of MVs in order to estimate these parameters. To achieve robust quantization, we suppress the influence of possibly inaccurate MVs by examining the smoothness of the MV field [23]. An MV that is very different from its neighbours, and therefore suspected to be inaccurate, will have less influence on the resulting quantization. A similar idea was studied in [14] in the context of

colour quantization. We first apply a 3×3 vector median filter to the MV field. Then, for each motion vector $\mathbf{MV}_j$, find the maximum Euclidean distance $D_{MAX,j}$ from its 8-adjacency neighbours and assign it the weight $W_j = \exp(-D_{MAX,j})$. Using these weights, we run a generalized Lloyd algorithm for vector quantization:

Step 1. Start with a single cluster (all MVs in the frame), compute its centroid $\mathbf{MV}_{cent}$ as

$$\mathbf{MV}_{cent} = \frac{\sum_j W_j \mathbf{MV}_j}{\sum_j W_j} \qquad \text{Equation 3.16}$$

then split it into two clusters by deriving two new centroids as $\mathbf{MV}_{cent} \pm \mathbf{MV}_{cent}/2$.

Step 2. Quantize all MVs in the frame into existing clusters using the nearest neighbor criterion. Then, for the i-th cluster C_i, update the centroid MV as:

$$\mathbf{MV}_{cent}^{C_i} = \frac{\sum_{\mathbf{MV}_n \in C_i} W_n \mathbf{MV}_n}{\sum_{\mathbf{MV}_n \in C_i} W_n} \qquad \text{Equation 3.17}$$

Step 3. Compute the weighted distortion of each cluster C_i:

$$WD^{C_i} = \sum_{\mathbf{MV}_n \in C_i} W_n \left\| \mathbf{MV}_n - \mathbf{MV}_{cent}^{C_i} \right\| \qquad \text{Equation 3.18}$$

Let C_k be the cluster with the maximum weighted distortion, and let X_{max}, X_{min}, Y_{max}, and Y_{min} be, respectively, the maximum and minimum horizontal and vertical component among the centroids. Split cluster C_k into two clusters with centroids $\mathbf{MV}_{cent}^{C_k} \pm \mathbf{P}$, where

$$\mathbf{P} = \left(\frac{X_{max} - X_{min}}{2(N-1)}, \frac{Y_{max} - Y_{min}}{2(N-1)} \right) \qquad \text{Equation 3.19}$$

and N is the total number of clusters prior to splitting.

Step 4. Repeat steps 2) and 3) until the total weighted distortion (sum of all WD^{C_i}) becomes less than a given threshold (in our experiments, 5% of its initial value in step 1), or the smallest cluster size becomes less than another threshold (in our experiments, 5% of the total MV field size).

The 5% thresholds on weighted distortion and cluster size have been chosen empirically based on our initial experiments, to balance the computational complexity and segmentation accuracy. Upon completion, a preliminary segmentation map is obtained: MVs in cluster C_j obtain the region label ω_j, which enables us to compute $m_{\omega_j}^X$, $m_{\omega_j}^Y$, $\sigma_{\omega_j}^X$, $\sigma_{\omega_j}^Y$ and $P(\omega_j)$.

3.4.3. Markov Random Field Motion Segmentation

For block j, based on Bayes' theorem, the posterior probability $P(\omega_j \mid \mathbf{MV}_j)$ is proportional to $P(\mathbf{MV}_j \mid \omega_j)P(\omega_j)$, so the Maximum A Posteriori (MAP) estimate of ω_j is given by:

$$\widehat{\omega}_j = \text{argmax}_{\omega_j} P(\mathbf{MV}_j|\omega_j)P(\omega_j) \qquad \text{Equation 3.20}$$

where $P(\mathbf{MV}_j \mid \omega_j)$ is computed as in Eq. 3.13 and $P(\omega_j)$ as in Eq. 3.15 and 3.16. The MAP segmentation for the entire MV field corresponds to maximizing:

$$\prod_j P(\mathbf{MV}_j|\omega_j)P(\omega_j) \qquad \text{Equation 3.21}$$

The final segmentation map is obtained using the method of Iterated Conditional Modes (ICM) [21], by iteratively solving Eq. 3.22 for each block in the frame. We use the ICM implementation from [15] (modified for MV segmentation instead of pixel segmentation), with six iterations. The final step is to identify small regions whose size is less than 2% of the total MV field, and group each block in those regions to the neighbouring large region with the closest centroid MV.

3.4.4. Boundary Refinement

A segmentation map obtained from the coarse segmentation (Fig. 3.22(a)) is block-based. Since real region boundaries rarely follow block boundaries, segmentation map must be refined. Figure 3.22(b) illustrates the boundary refinement procedure. Based on motion consistency along the coarsely segmented regions, we first identify the blocks that likely contain region boundaries, and then apply a region growing procedure to obtain pixel-wise boundaries using features such as edges and colour.

The boundary refinement process consists of three steps: boundary block identification (Fig. 3.24 (a)-(b)), edge detection (Fig. 3.24 (c)-(d)), and interior region growing (Fig. 3.24 (e)-(f)). Boundary blocks are identified in the segmentation map from Section 3.4.3 using the Region Motion Deviation (RMD) map. The RMD value I_j^R of $\mathbf{MV}_j$ within region R is the normalized deviation of $\mathbf{MV}_j$ from the centroid MV of region R:

$$I_j^R = 255 \times \left({D_j^R}/{D_{max}^R} \right), \qquad \text{Equation 3.22}$$

where

$$D_j^R = \left\| \mathbf{MV}_{cent}^R - \mathbf{MV}_j \right\|, \qquad D_{max}^R = \max_j \left(D_j^R \right). \qquad \text{Equation 3.23}$$

A two-pass procedure is employed to classify a block as either a *boundary block* or *interior block*. In the first pass, we scan all the blocks in the raster scan order and for each block we check its East (E), South (S), and South-East (SE) neighbouring blocks, if available. If any of these blocks belong to a different region than the one the current block belongs to, we compare the RMD values of all four blocks (current, E, S, SE), and label the block with the highest RMD value as a boundary block. In the second pass, we seek to extend the boundary to be at least 2 blocks (16 pixels) wide, to improve the chance that the real region boundaries lie within boundary blocks. To do this, we check the 4 adjacent neighbours of all boundary blocks found so far and check if they have at least one horizontal (vertical) neighbour classified as a boundary block. If not, we label

the horizontal (vertical) neighbour with the higher RMD value as a boundary block. At the end, all blocks not classified as boundary blocks are labelled as interior blocks.

A Canny edge detector on the Y-component is used to identify edges within boundary blocks as shown in Fig. 3.24(c). Then, interior regions are grown towards each other via morphological erosion of the boundary blocks using a 3×3 structured element. During the process, the structured element is not allowed to cross an edge. Hence, this restricted erosion will move the interior region boundaries up to the nearest edge(s). In this process some boundaries of neighbouring interior regions may meet, in which case the pixel-wise boundary between these regions is identified. In other cases boundaries do not meet due to a complicated edge pattern between them, so we further employ region growing based on colour, as in [27], to finalize region boundaries.

In Fig. 3.24, we illustrate the refinement procedure on frame #22 from *Flower Garden*. Figures 3.24 (a)-(b) show boundary block identification, Figs. 3.24 (c)-(d) show detected edges in boundary regions and Figs. 3.24 (e)-(f) show interior region growing. In this example, the comparison is made between the segmentation produced by the algorithm from our previous work [75] that incorporates GMC (Fig. 3.24 (a), (c), (e)), and the one produced in our previous work [73] (Fig. 3.24 (b), (d), (f)), which does not include GMC. In both cases the tree trunk is well-segmented, but the method from [73] ends up with a higher number of regions in the background due to its lack of GMC.

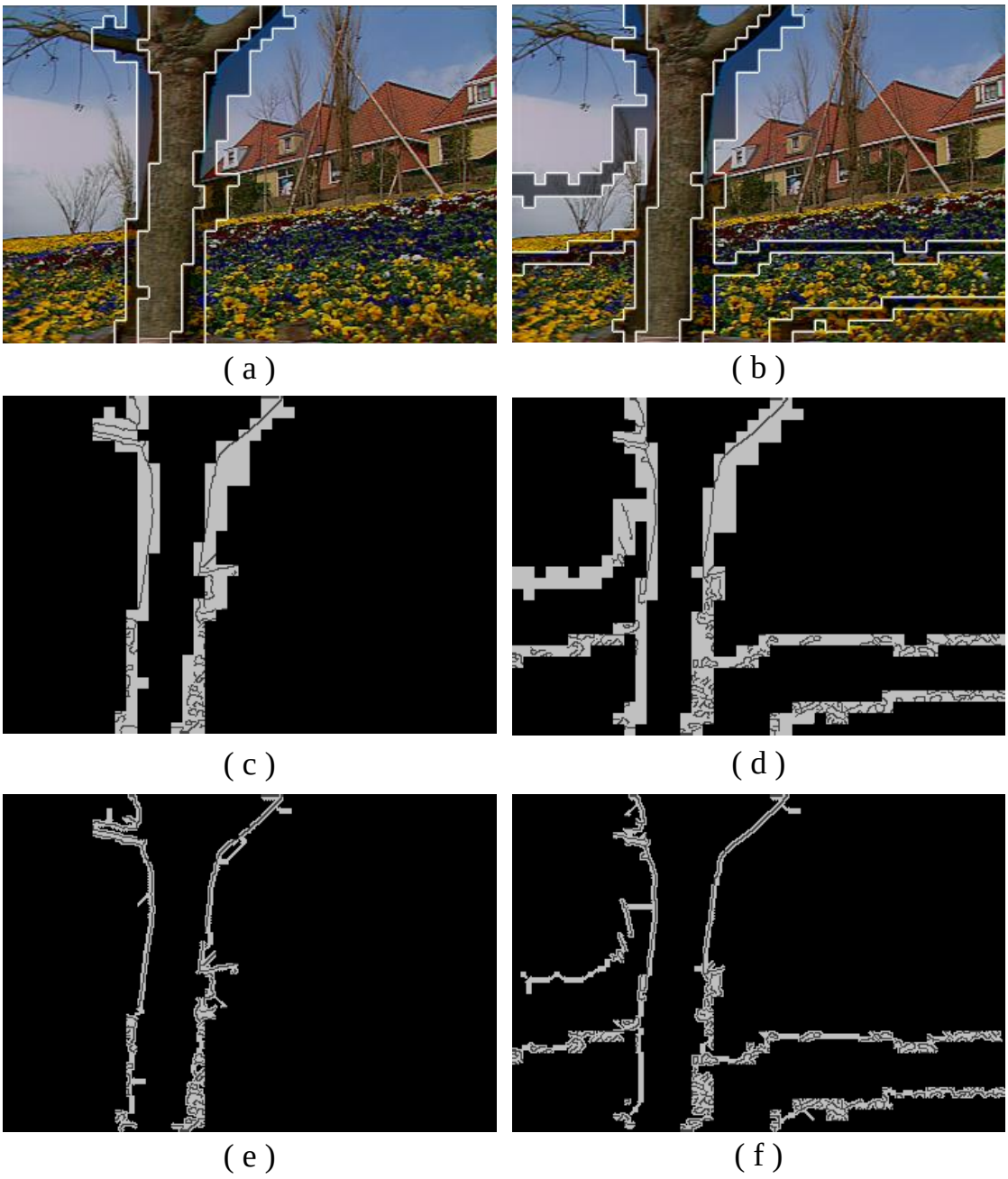


Figure 3.24: [Top]: Coarse segmentation and identified boundary blocks, [Middle]: Initial boundary regions and edges within them, [Bottom]: result of interior region growing. Segmentation with GMC is employed on the left, while segmentation without GMC is employed on the right.

3.4.5. Evaluation

The MRF segmentation algorithm has been tested on a variety of standard sequences with different motion characteristics. The sequences are CIF (352×288) and SIF (352×240) resolution with a frame rate of 30 frames per second. In this work we use the XviD MPEG-4 codec (http://www.xvid.org/) for compression, with the IPPP GOP structure, at 512 kbps. We point out that the segmentation framework is generic and easily adapted to other video compression standards. The MVs extracted from the bitstream are normalized to form a uniformly sampled MV field, where each MV corresponds to an 8×8 block.

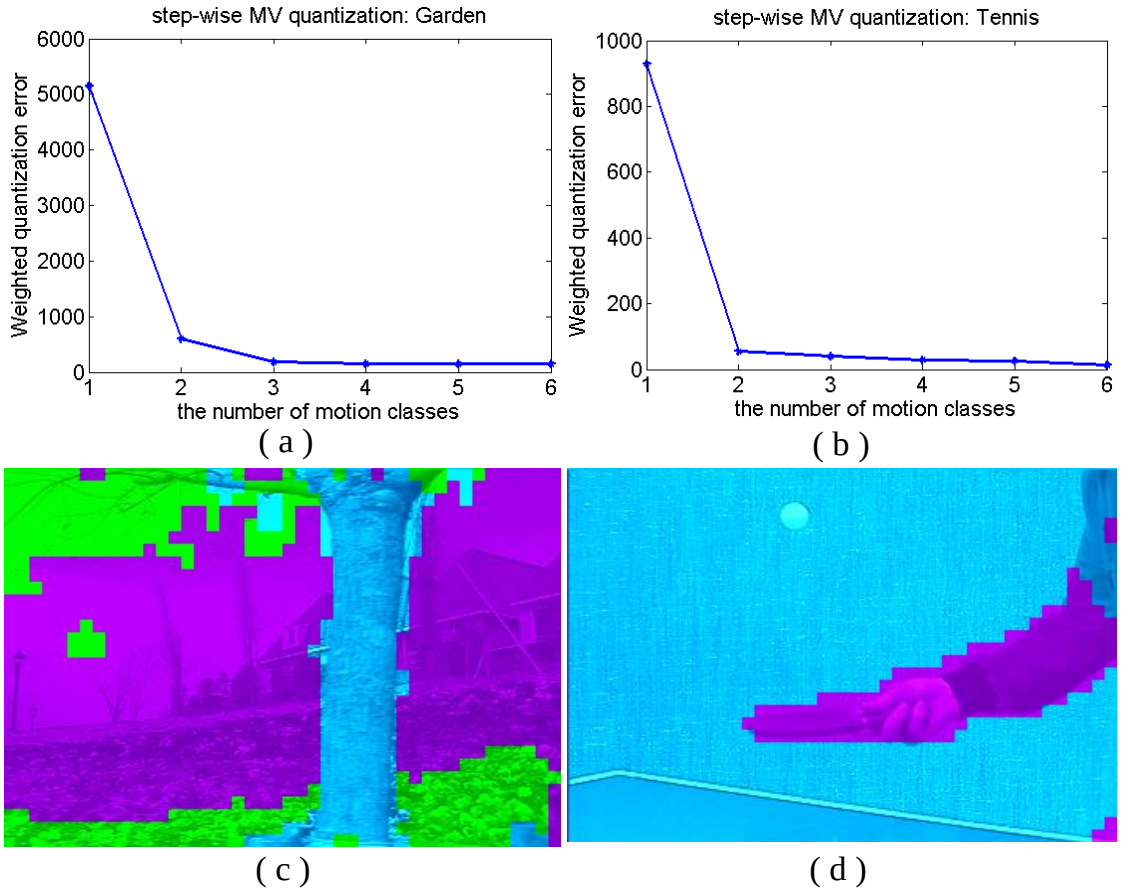


Figure 3.25: (a) and (b): weighted quantization error vs. the number of motion classes, (c) and (d): the corresponding segmentation map after MV quantization, where segments are distinguished by different colours.

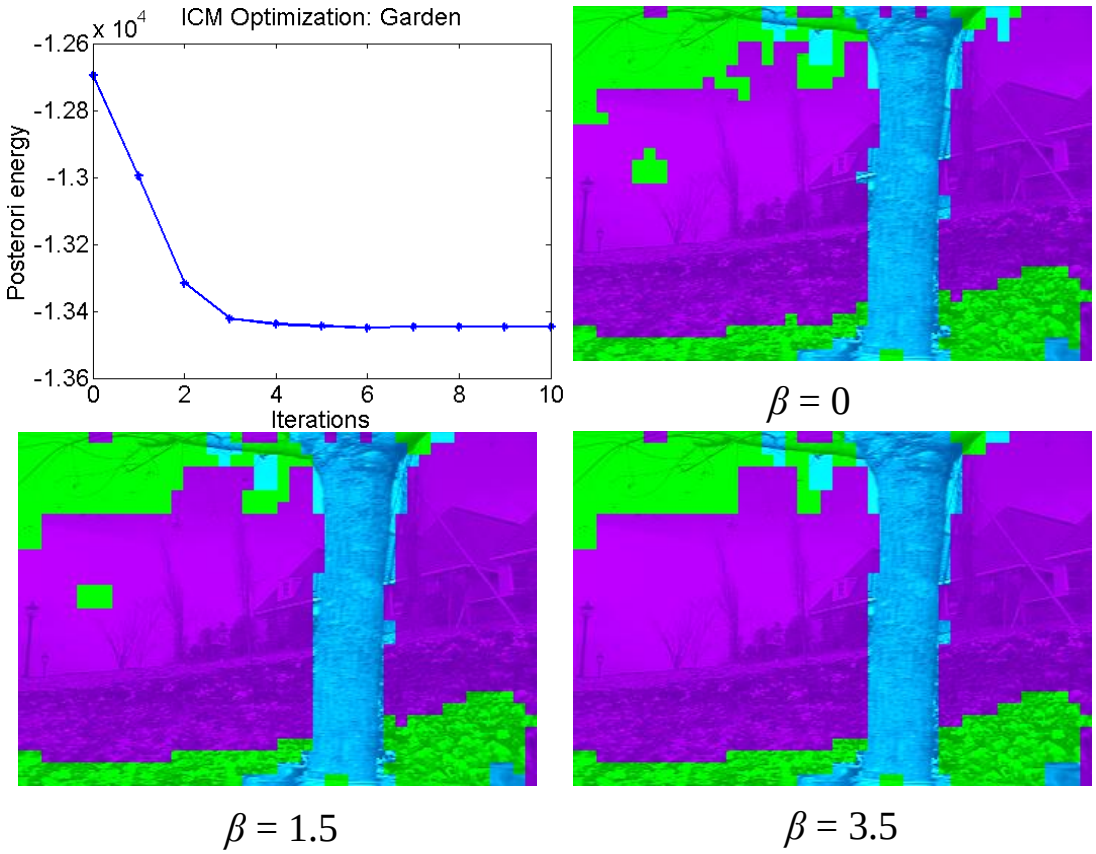


Figure 3.26: [Top Left]: Posteriori energy vs. iterations; Other figures: MRF segmentation with $\beta \in \{0, 1.5, 3.5\}$.

We first evaluate MV quantization as a way to determine the number of MRF classes and to provide the initial segmentation map. Figures 3.25(a) and (b) show how the weighted quantization distortion changes as a function of the number of classes on sample frames from *Flower Garden* (frame #2) and *Table Tennis* (frame #4). Figure 3.25(a) indicates that three classes seem to be appropriate for frame #2 of *Flower Garden*, while Fig. 3.25(b) indicates that two classes are appropriate for frame #4 of *Table Tennis*. The corresponding initial segmentation maps are shown in Figs. 3.25(c) and (d), respectively. These initial segmentation maps enable us to calculate the means and variances of horizontal and vertical MV components within each region.

Next, we evaluate MRF segmentation, especially the number of ICM iterations and the role of parameter β in Eq. 3.15, which influences the spatial structure of the

MRF. In the top left part of Fig. 3.26 we show the posteriori energy (the sum of potentials in Eq. 3.15 of all cliques in the field) vs. the number of iterations of ICM implementation from [15] when $\beta = 3.5$. The graph indicates that 4-6 iterations are sufficient, as suggested in [15]. Hence, we used 6 iterations in all our experiments.

The rest of Fig. 3.26 shows the segmentation of frame #2 of Flower *Garden* obtained by setting β to 0, 1.5, and 3.5, respectively. When $\beta = 0$, no spatial constraints are imposed on the MRF, so the segmentation does not change from its initial layout obtained by MV quantization (Fig. 3.25(c)). As β increases, neighbouring blocks are more likely to be in the same region, so region boundaries end up being more compact. Our experiments indicate that $\beta = 3.5$ provides a good balance between boundary compactness and segmentation accuracy, so we use this value in the remaining experiments.

In Fig. 3.27(a), we illustrate the final MRF segmentation of frame #2 of *Flower Garden* (after merging blocks from small regions to neighbouring regions) and in Fig. 3.27(b) we show the boundary refinement results. For comparison we also show the results from four other state-of-the-art segmentation algorithms — [14], [67], [69] and [27]. Figure 3.27(c) shows the segmentation result using the algorithm from [14], which is image-based and does not use motion information and thus results in over-segmentation. This problem has been mitigated to some extent by the method proposed in [27], shown in Fig. 3.27(d), which utilizes *k*-means clustering and motion consistency. However, the scene is still over-segmented. Figure 3.27(e) shows the result of using the method from [67], which is based on MRF with two motion classes (background and foreground). The result is an under-segmented scene with part of the background (garden) included in the same segment as the foreground (tree trunk). Finally, Fig. 3.27(f) shows the segmentation result from [69], which is a MV-based approach using the Expectation Maximization algorithm on a dense MV field. This method ends up with the same number of motion classes as ours, but these motion classes (segments) are less compact than in our case and some are not even spatially connected. A few other segmentation results of our method are shown in Fig. 3.28.

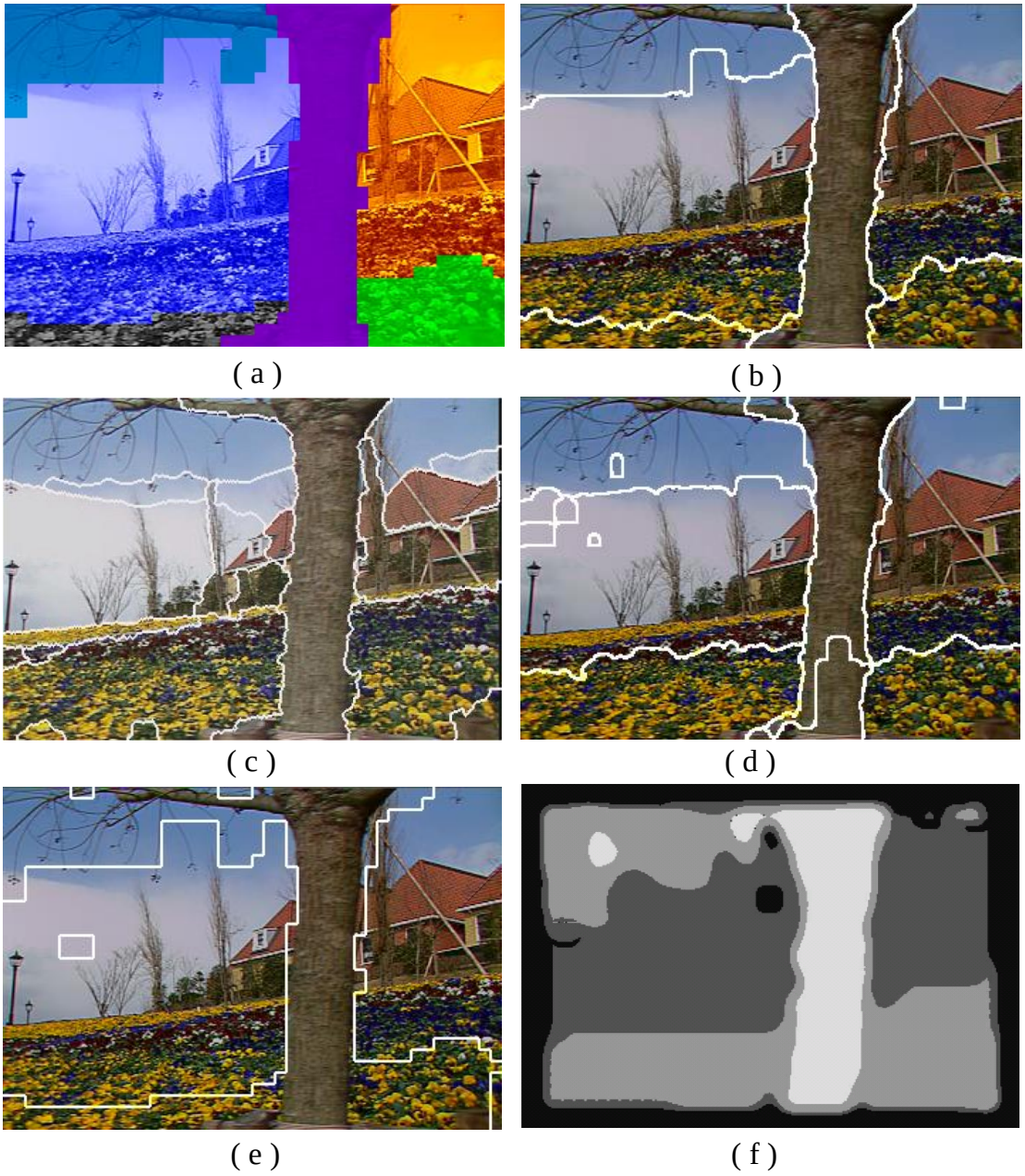


Figure 3.27: (a): Coarse MRF segmentation, (b):boundary refinement. (c, d, e, f): segmentation result from Ref. [14], [27], [67] and [69], respectively.

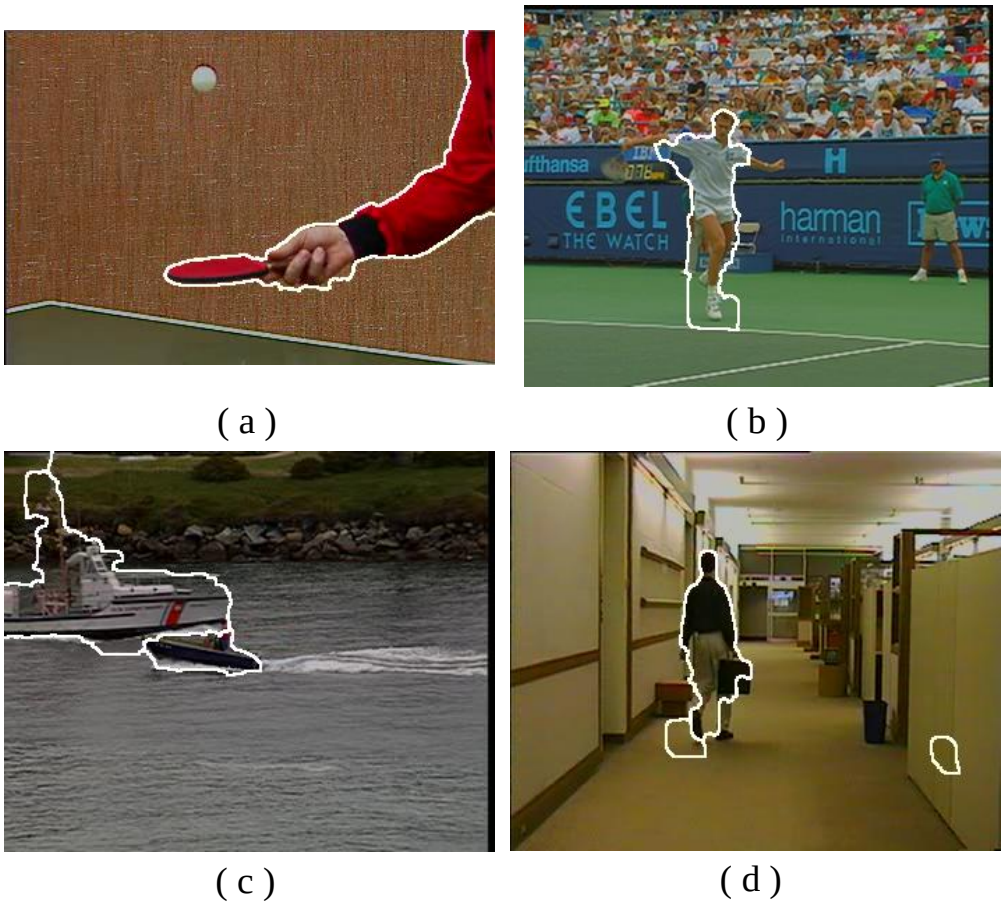


Figure 3.28: Final segmentation results from sequences: (a) *Table Tennis* (frame #5), (b) *Stefan* (frame #31), (c) *Coastguard* (frame #40), and (d) *Hall Monitor* (frame #50).

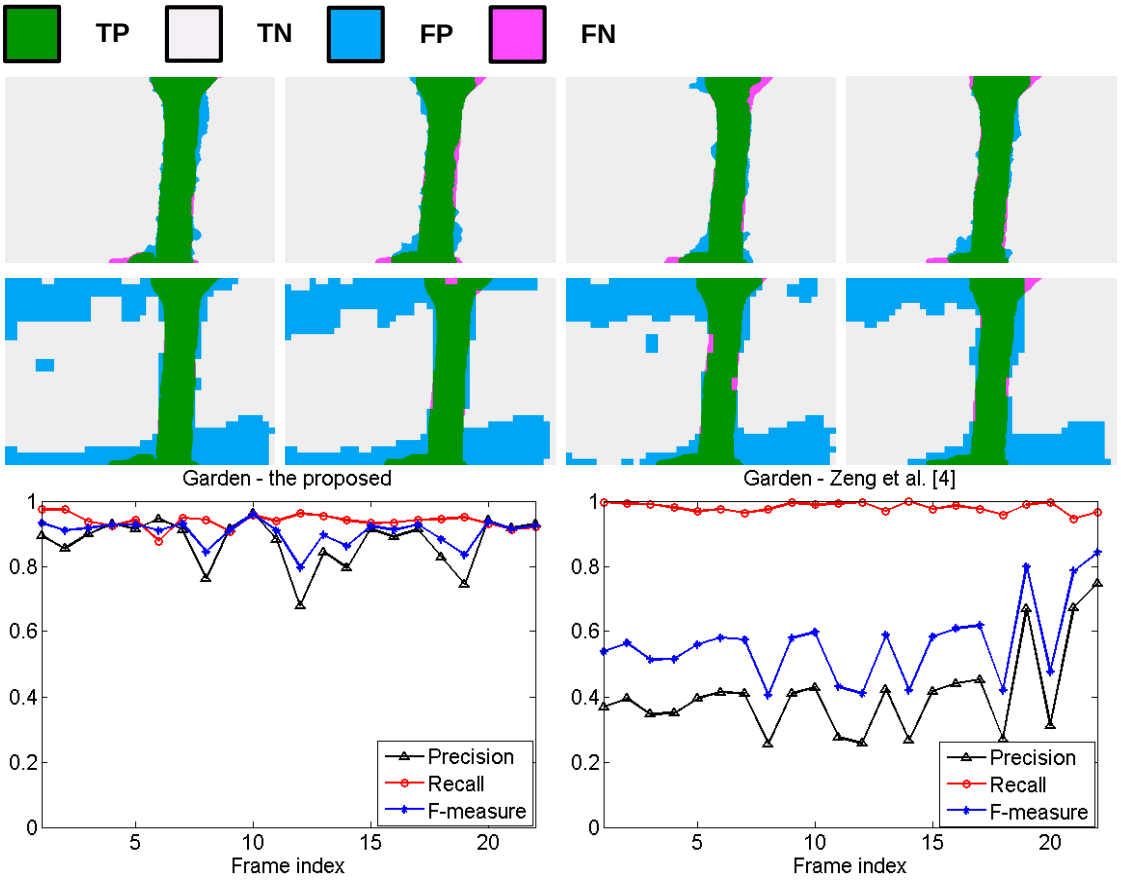


Figure 3.29: Quantitative evaluation – *Flower Garden.* [Top]: the proposed method, frame #1, #3, #5, #7, from left to right, [Middle]: corresponding segmentation using method from [4], [Bottom]: the quantitative evaluation for the proposed method (left) and method from [4] (right).

In addition to the visual results above, we provide a quantitative evaluation of our method, using the manually segmented sequences *Flower Garden* and *Table Tennis.* We test how accurately the fastest moving objects (tree trunk in *Flower Garden*, player's hand and ball in *Table Tennis*) can be segmented. By counting the True Positive (TP) pixels, True Negative (TN) pixels, False Positive (FP) pixels, and False Negative (FN) pixels, we can compute several quantities for measuring segmentation accuracy: Precision = TP / (TP + FP), Recall = TN / (TN + FN), and F-measure as the harmonic mean of Precision and Recall. In terms of these quantities, we compare our method and the method from [67], which is the latest work addressing MRF motion segmentation in

block-based compressed video. In our implementation, 8×8 uniformly sampled MV field is used.

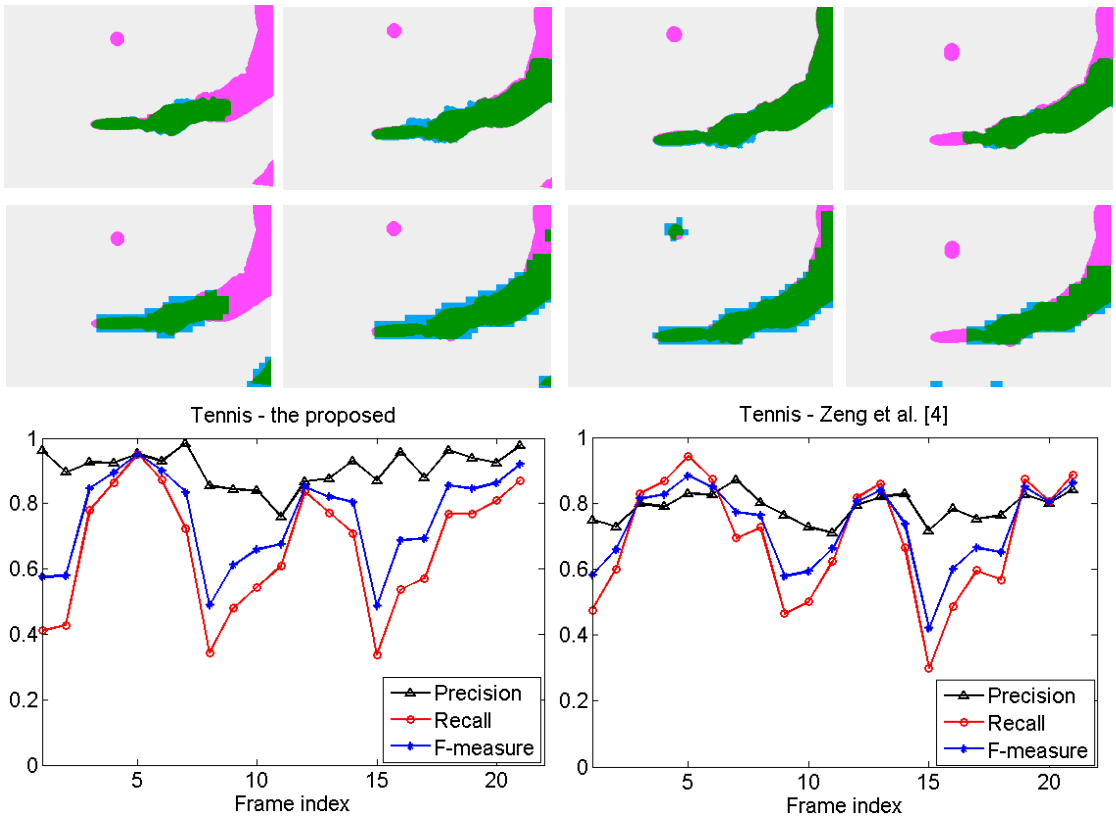


Figure 3.30: Quantitative evaluation – *Table Tennis.* [Top]: the proposed method, frame #1, #3, #5, #7, from left to right, [Middle]: corresponding segmentation using the method from [4], [Bottom]: the quantitative evaluation for the proposed method (left) and the method from [4] (right).

The top and middle rows in Figs. 3.29–3.30 show the segmented objects in frames #1, #3, #5, and #7 extracted by our method and the one from [67]. The TP, TN, FP, and FN pixels are also shown. The last row in both figures shows the quantitative measures for the first 25 frames of *Flower Garden* and *Table Tennis*, while their averages are listed in Table 3.5. For *Flower Garden*, the method from [4] has an average precision of 0.41 due to under-segmentation (background pixels included in the foreground, shown as blue pixels in Fig. 3.29), while our method maintains a much

higher precision of 0.86. The performance of the two methods is more similar on *Table Tennis*, where the assumption made in [67] about two motion classes (foreground and background) is more appropriate. Nonetheless, our boundary refinement yields more accurate boundaries, which again leads to higher precision (0.91 vs. 0.79). Finally, note that our segmentation method has a reasonably low complexity. On a standard desktop PC with an Intel Pentium CPU at 3.0 GHz and 2 GB of RAM, on a CIF sequence, motion segmentation (in MATLAB) takes on average about 80ms per frame, while boundary refinement (in C/C++, Visual Studio 7) takes about 20ms.

Table 3.5: Average precision, recall, and f-measure.

Sequence	*Flower Garden*		*Table Tennis*	
	Proposed	**Ref. [67]**	**Proposed**	**Ref. [67]**
Precision	0.86	0.41	0.91	0.79
Recall	0.94	0.98	0.67	0.69
F-measure	0.90	0.56	0.75	0.72

4. Joint Approach to Global Motion Modelling and Segmentation

In Chapter 2 and 3, we have discussed global motion modeling and object segmentation as two separate tasks, however, in many content-based video processing systems, the presence of moving objects limits the accuracy of global motion estimation (GME). On the other hand, the inaccuracy of global motion parameter estimates affects the performance of motion segmentation. In this chapter we are going to describe a framework for simultaneous global motion modelling and segmentation in compressed video. Some of the material in this chapter can be found in our earlier work [126]. The proposed procedure starts with removing MV outliers from the MV field and then performs GME to obtain an estimate of global motion parameters. Using these estimates global motion is removed from the MV field and moving region segmentation is performed on this compensated MV field. MVs in the moving regions are treated as outliers in the context of GME in the next round of processing. Iterating between GME and motion segmentation helps improve both GME and segmentation accuracy.

4.1. Introduction

4.1.1. GME and Motion Segmentation

Global motion estimation (GME) and motion segmentation are two widely used techniques in video coding [4], computer vision [5-6] and content-based video analysis [7-9]. GME estimates motion caused by camera movement in a video sequence [1] and can be done in either the pixel domain [2] [4] [8] [10] or compressed domain [3] [11-13] [29]. Compressed-domain approaches are computationally less demanding, since they utilize block-based MVs from the compressed bit stream. However, their accuracy may also suffer due to outliers in the MV field, caused either by imperfect motion estimation at the encoder or by objects whose motion is different from the camera motion. Hence,

identifying and removing outliers is one of the main challenges of compressed-domain GME. In the approach proposed here motion segmentation is the cornerstone of outlier removal.

Combined motion estimation and segmentation have been studied previously in the context of optical flow estimation. For example, in [33] a 6-parameter affine motion model is estimated for each segment in the scene at the same time as segmenting the optical flow field. The approach in [34] generalizes [33] and several other related methods by introducing the residual motion field to account for model errors, in addition to the 6-parameter affine motion field for each segment. In both these works, the flow is modelled as a Markov Random Field (MRF) and Bayesian segmentation is used to isolate regions that appear to move differently from each other.

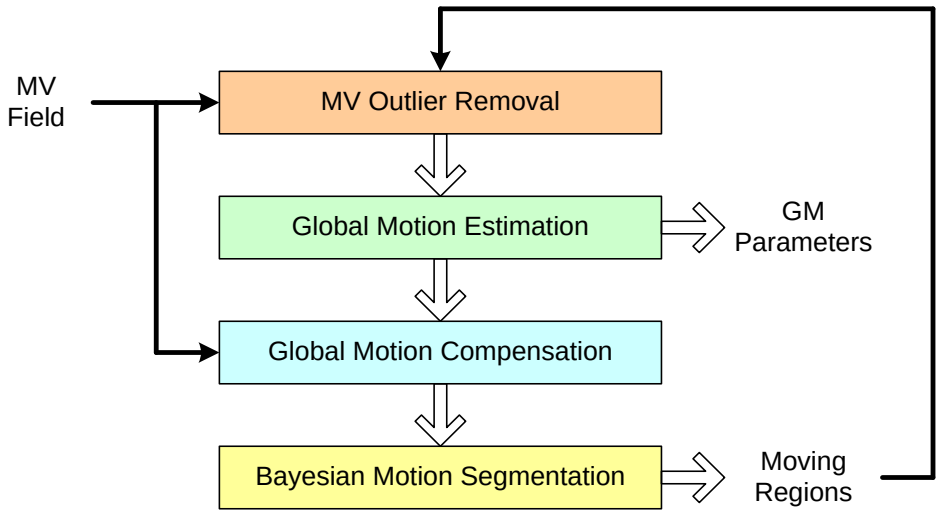


Figure 4.1: The system diagram of the joint approach to compressed-domain global motion estimation and motion segmentation. The framework offers combined GM parameters and video moving object information.

Although motion estimation and segmentation have been considered jointly in the context of optical flow estimation, as discussed above, in the literature on GME itself motion estimation [2-13] and object segmentation [14-19] are usually treated as two separate topics. Notable exceptions, where some connection between GME and

segmentation is made, include [8] and [20]. In [20], Liu *et al*. proposed a segmentation framework in the compressed domain, where GM compensation is employed to obtain MV residuals prior to segmentation. In their approach GME itself was conducted without moving object removal from the MV field. Qi *et al*. [8] developed a pixel-domain GME framework for video object segmentation, where object information is obtained by performing GM compensation in the pixel domain and used to predict outlier blocks for GME in the next frame.

4.1.2. Proposed Joint Approach to GME and Motion Segmentation

The proposed joint global motion modelling and segmentation is shown in Fig. 4.1, where GME and motion segmentation are tightly coupled. Motion segmentation helps remove MV outliers from the input MV field, leading to more accurate GME. Global motion is then used to perform GM compensation, which leads to more accurate motion segmentation on the GM-compensated MV field.

Motion segmentation in this joint approach uses the MRF model and the Bayesian approach, which is similar to [6] [33] and [34] but differs in a number of ways. First, our main goal is GME, that is, the estimation of motion caused by camera movement. This motion is usually associated with the background, which is approximated as a flat surface far from the camera. Hence, for this purpose, an 8-parameter perspective motion model [6], which we use here, is more appropriate than the 6-parameter affine model used in [33] and [34]. Also, while the methods in [33] and [34] operate on the raw video in the pixel domain, our method is developed for compressed video to operate directly on MVs from the compressed bit stream, resulting in much lower complexity. In addition, motion segmentation in our work is performed on the global motion-compensated MV field, that is, the field from which the estimated camera motion has been removed. By contrast, [33] and [34] perform motion segmentation directly on the observed optical field that includes camera motion bias, which is known to reduce segmentation accuracy. Finally, the main purpose of motion segmentation in our work is to remove the MVs that belong to individual moving objects (i.e. outliers) and thus improve the accuracy of GME, while motion estimation for individual objects is not explicitly considered. From this point of view, the approach presented here can be utilized to initialize methods like [33] or [34] (both require

initialization), which could then supply local motion estimation and segmentation refinement.

Also notice important differences between our approach and the combined GME and object segmentation method from [8] as follows: 1) reference [8] uses a 6-parameter affine model for the global motion, while we use the more appropriate 8-parameter perspective model; 2) reference [8] uses GME and segmentation in cascade, while we use them iteratively; 3) the method in [8] operates in the pixel domain, while our method works in the compressed domain, i.e. directly on MVs; 4) reference [8] performs segmentation by differencing the transformed frame from the reference frame, while our method uses MRF-based motion segmentation.

Overall, the proposed approach couples object segmentation and GME on the block-based MV field and introduces several contributions to GME research. First, motion segmentation offers a principled method to distinguish outlier MVs caused by moving objects or objects close to the camera, and leads to fast convergence of GM parameter estimates. Second, motion parameters are fed back to the segmentation process to compensate for global motion, thus mitigating segmentation problems found in scenes with a moving camera. Third, the approach is applicable to any video bit stream compressed by a block-based standards-compliant encoder (e.g. MPEG-2/4, H.263 and H.264), since the MV field is the only information required. The proposed method has a higher degree of flexibility and portability compared to some compressed-domain approaches that rely on codec-specific information [16] [31], such as DCT (transform) coefficients and MB types.

Not all outliers in the MV field can be identified by motion segmentation. As discussed in Chapter 2, there are two types of outliers present in the MV field: **Type-1** outliers, caused by imperfect motion estimation at the encoder, and **Type-2** outliers, caused by moving objects or objects close to the camera. The former differs from the latter due to their noise-like characteristics. Therefore, we detect **Type-1** outliers by examining their similarity to the neighbouring MVs, while **Type-2** outliers are isolated using MRF-based segmentation.

The rest of this chapter is organized as follows. In Section 4.2, we describe MV outlier removal and model-fitting (i.e. regression) methods used in the proposed GME. Then, in Section 4.3 we present the Bayesian motion segmentation on a GM-compensated MV field. Finally, in Section 4.4 we provide experimental results on motion segmentation and GME.

4.2. MV-based Global Motion Modelling

Reference [3] summarizes four popular 2D motion models for global motion (translational, geometric, affine and perspective). The 8-parameter perspective projection model is the most general of these and is adopted in our work for modelling global motion. This model accurately represents the 2D projection of the 3D rigid motion of a planar surface [6]. Therefore, we adopt this model to represents the motion of the background, which is often far from the camera and can be approximated by a planar surface. The perspective model is parameterized by an 8-dimensional vector $\mathbf{m} = [m_0, \cdots, m_7]$. Given (x, y) and (x', y') as the coordinates in the current and the reference frame, respectively, the perspective transformation is defined as in Equation 2.3.

As in Chapter 2, let MV^X and MV^Y denote, respectively, the horizontal (*X*-) and vertical (*Y*-) component of a motion vector $\mathbf{MV} = (MV^X, MV^Y)$. Then MV^X and MV^Y for the MV of the block centered at pixel (x, y) in the current frame, corresponding to the motion model $\mathbf{m}$ are given by Equation 2.7.

The importance of both Type-1 and Type-2 outliers removal to accurate GME has been addressed through various objective and subjective evaluation in Chapter 2. Basically, in the literature there are three main approaches to outlier removal for GME: 1) RANdom SAmple Consensus (RANSAC), 2) robust estimation using a regression method such as gradient descent [3] or least squares [11], and 3) explicit MV outliers filtering. As discussed in Chapter 2, we found that MV outlier filtering is the fastest of the three approaches. It tends to work very well on Type-1 outliers, since these "noisy" MVs tend to be different from their neighbouring MVs, but it may run into problems when faced with Type-2 outliers. RANSAC tends to work well on Type-2 outliers, but its performance is not as good on Type-1 outliers. Meanwhile, iterative robust estimators

tend to work well on both Type-1 outliers and Type-2 outliers, as long as their percentage in the MV field is not large. These findings were used as a basis for developing a joint outlier removal strategy, elaborated on in this chapter.

4.2.1. Type-1 MV Outlier Removal

MVs that come from a given motion model (i.e., a given vector of motion parameters $\mathbf{m}$) usually exhibit fairly strong spatial correlation [28]. Hence, in most cases, Type-1 MV outliers can be reliably detected by checking their similarity to their neighbouring MVs. In this work, we use the first filter from the filter cascade in Chapter 2, section 2.2.4 (and our earlier work [28]) to remove Type-1 outliers. This filter examines the magnitude and phase difference between an MV and its 8-adjacency neighbours and then removes a prescribed fraction of the worst-fitting MVs from the MV field as outliers. In our experiments, this fraction is set to be 0.15.

4.2.2. Type-2 MV Outlier Removal

Type-2 outliers tend to be similar to their neighbouring MVs that belong to the same object. Hence, outlier removal based on spatial dissimilarity does not work well on these MVs. In this section, we develop an iterative motion segmentation method to detect and remove Type-2 MV outliers, which is applied starting from the second P-frame and onwards (in the first P-frame only Type-1 outliers are removed). In the first iteration, the moving region segmentation map is predicted from the segmentation result in the previous frame; in subsequent iterations the segmentation map of the current frame from the previous iteration is used.

As before, let $\mathbf{m}_t$ be the vector of estimated GM parameters at a particular iteration in frame t, and $\mathbf{MV}_{x,y}(t)$ be the MV of the block centered at pixel (x, y) in frame t. Then the global motion can be compensated from $\mathbf{MV}_{x,y}(t)$ by:

$$\mathbf{MV}_{x,y}^{res}(t) = \mathbf{MV}_{x,y}(t) - \mathbf{MV}_{x,y}(t; \mathbf{m}_t) \qquad \text{Equation 4.1}$$

where $\mathbf{MV}_{x,y}^{res}(t)$ is the compensated MV whose block is centered at pixel (x, y) in frame t and $\mathbf{MV}_{x,y}(t; \mathbf{m}_t)$ is obtained as in Eq. 2.3 and 2.7. After GM compensation, we run

Bayesian motion segmentation on the compensated MVs as described in Section 4.3. The segmentation result indicates where the moving regions are in the current frame, so MVs in these regions are identified as Type-2 MV outliers and removed prior to GME in the next iteration. Moreover, once the segmentation in frame t converges, the location of the Type-2 outliers in frame $t+1$ can be predicted from their MVs in frame t, as shown in Fig. 4.2. This is different from the approach in [8], where the prediction of outliers in the next frame does not take their motion into account. If $\mathbf{MV}_{x,y}(t)$ is declared a Type-2 outlier, then the location (x_{t+1}, y_{t+1}) of the center of the corresponding block in frame $t+1$ is predicted as:

$$(x_{t+1}, y_{t+1}) = (x_t, y_t) - \mathbf{MV}_{x,y}(t) \qquad \text{Equation 4.2}$$

The block centered at (x_{t+1}, y_{t+1}) might overlap several blocks in frame $t+1$, so we declare the block with the largest overlap as the predicted outlier block. In the first iteration of GME in frame $t+1$, all MVs associated with these predicted outlier blocks are removed prior to GME. After this initial GME and GM compensation, motion segmentation is used to detect Type-2 MV outliers more precisely in subsequent iterations.

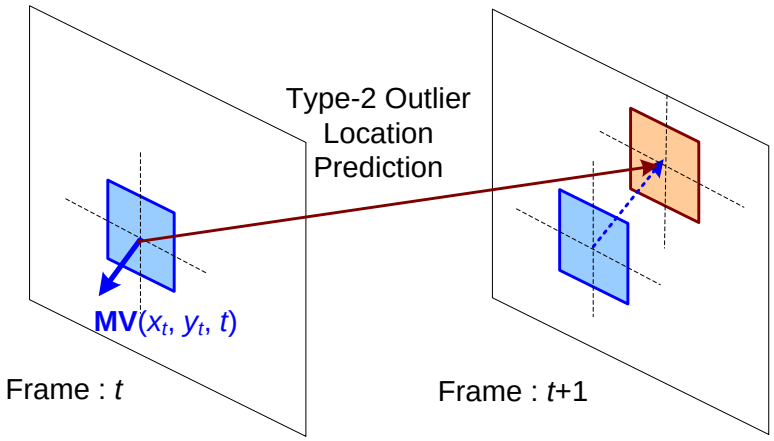


Figure 4.2: Type-2 outlier location prediction.

4.2.3. Global Motion Parameter Estimation

Once the outliers are removed, parameter vector $\mathbf{m}_t$ is estimated from the remaining MVs in frame t. The MV at location (x, y) in the frame corresponding to a given $\mathbf{m}_t$ is $\mathbf{MV}_{x,y}(t; \mathbf{m})$, and can be computed according to Eq. 2.3 and 2.7. Meanwhile, the actual MV at that location is $\mathbf{MV}_{x,y}(t)$. The goal of GME is to find $\mathbf{m}_t$ that minimizes the difference between $\mathbf{MV}_{x,y}(t; \mathbf{m})$ and $\mathbf{MV}_{x,y}(t)$. Squared error is the typical error criterion used in GME, so the problem can be formulated as follows

$$\mathbf{m}_t = \arg\min_{\mathbf{m}} \sum \left\| \mathbf{MV}_{x,y}(t) - \mathbf{MV}_{x,y}(t; \mathbf{m}) \right\|^2 \qquad \text{Equation 4.3}$$

where the sum goes over all locations (x, y) with inlier MVs (i.e. those MVs that have not been declared outliers). The process of finding $\mathbf{m}_t$ in Eq. 4.3 is referred to as model fitting, or regression. The two most popular regression methods for GME are gradient descent (GD) [3] and the least-squares (LS) approach [11] [29]. We used both methods in our experiments.

The GD approach in [3] is based on the Newton-Raphson method. The gradient and the Hessian of the error in Eq. 4.3 are found by differentiating the error with respect to the components of $\mathbf{m}_t$, after which an update to the current vector $\mathbf{m}_t$ is computed so that the error is reduced. The process is repeated until the error reduction becomes small enough. On the other hand, the LS approach involves formulating the problem in Eq. 4.3 as an over-determined linear system of the form $\mathbf{A}^T\mathbf{A}\mathbf{m} = \mathbf{A}^T\mathbf{b}$ [29], where $\mathbf{A}$ is a $2N \times 8$ matrix containing motion information, $\mathbf{b}$ is a $2N$-dimensional vector containing transformed coordinates from Eq. 4.1 and N is the total number of MVs. The problem is solved by computing a pseudo-inverse of $\mathbf{A}^T\mathbf{A}$, usually via singular value decomposition [29]. The particular LS method in [11] involves iterative outlier rejection through a robust M-estimator and has been termed LSS-ME.

These processes — Type-1 and Type-2 MV outlier removal, GME, GMC and motion segmentation — form one iteration of joint GME and segmentation (Fig. 4.1) and the procedure iterates until GME parameters converge.

4.3. Bayesian Motion Segmentation

After compensating for the global motion in the MV field, we perform Bayesian motion segmentation, as shown in Fig. 4.3. The process involves two steps: 1) MV clustering, which generates a preliminary segmentation map, as well as Markov Random Field (MRF) priors (the number of MRF classes and their parameters), and 2) Bayesian MV segmentation, which finds foreground moving regions.

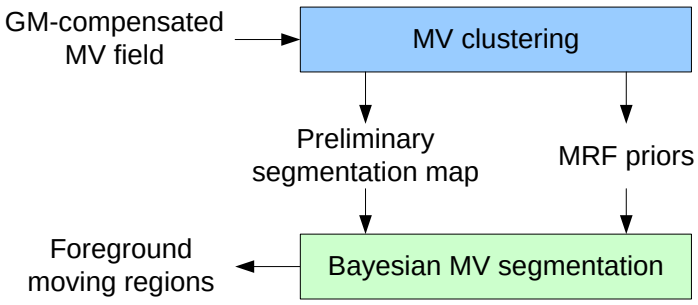


Figure 4.3: Overview of Bayesian motion segmentation.

4.3.1. Markov Random Field Motion Model and Parameter Estimation

As in Section 3.4, motion vectors $\mathbf{MV} = (MV^X, MV^Y)$ within a given moving region ω follow a conditional distribution $P(\mathbf{MV} \mid \omega)$, while region labels ($\omega$'s) follow a 2D MRF distribution based on a given neighbourhood system. The goal is to infer region labels (ω's) from the observed MV field. The details of the MRF motion model and MRF parameter estimation elaborated in Section 3.4.1.

While MV clustering provides a fairly robust way to determine the number of moving regions and their approximate statistics, other parameters, such as the potential β in Equation 3.12, are more difficult to set. In [19], the authors have used expectation-maximization (EM) to set the potential value. In this work, however, we felt that adding EM into the segmentation-GME loop would lead to a prohibitively large amount of computation. Hence, we used an empirically determined value $\beta = 3.5$ (please see

Section 3.4.5 for details). While this value is not necessarily optimal, it is sufficiently good to illustrate the advantages of the proposed method.

Upon completion, we compare the *X*- and *Y*-components of the MV centroid of each region with thresholds computed from all MVs in the GM-compensated MV field, as explained below. If each component is lower than the corresponding threshold the region is considered a part of the background. Otherwise, the region is considered a part of a foreground moving object and declared a Type-2 outlier.

The thresholds are computed as follows. Let $\mathbf{MV}_i^{res,X}$ and $\mathbf{MV}_i^{res,Y}$ be the *X*- and *Y*-component of the i-th MV in the GM-compensated MV field, respectively. Thresholds TH^X and TH^Y are computed as:

$$TH^X = \kappa \cdot \sqrt{\frac{1}{N} \cdot \sum_i \left(\mathbf{MV}_i^{res,X}\right)^2} \qquad \text{Equation 4.4}$$

$$TH^Y = \kappa \cdot \sqrt{\frac{1}{N} \cdot \sum_i \left(\mathbf{MV}_i^{res,Y}\right)^2} \qquad \text{Equation 4.5}$$

where $\kappa = 1.2$ in our experiments.

4.4. Evaluation

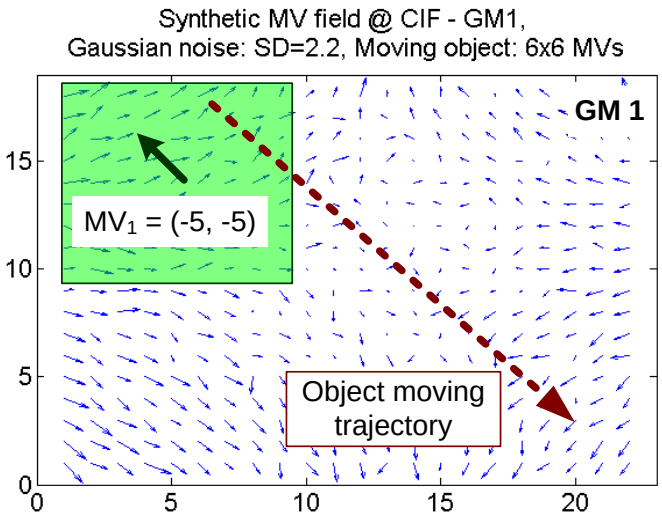
In this section, we present simulation results to evaluate the performance of the proposed joint GME and motion segmentation. We compare our method with other MV-based GME approaches: robust iterative MV-based GME (GD [3] and LSS-ME [11]), GME with MV outlier rejection filter from [23], and GME using RANdom SAmple Consensus (RANSAC) [6] [22]. Evaluation is conducted on both synthetic MV fields and real video sequences. In the case of GD, 65% of the best-fitting MVs are used as inliers to update GM parameters at each iteration, as in [3]. In the case of LSS-ME [11], the weights of outliers are set to zero and parameter *C* is set to 2 in our experiments.

4.4.1. Evaluation on Synthetic MV fields

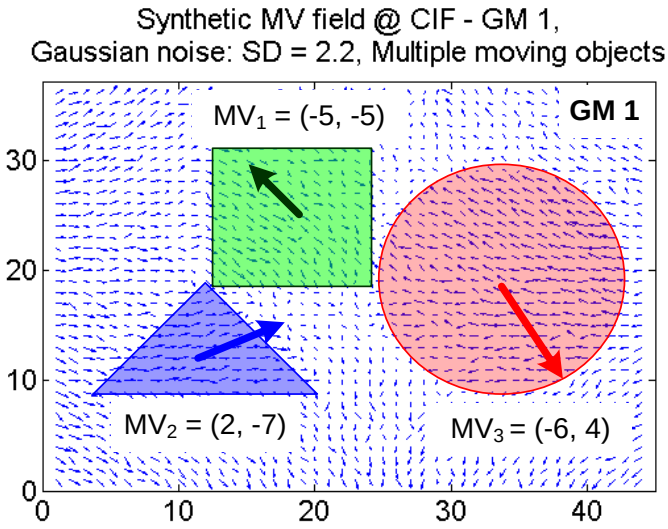
The synthetic MV fields are synthesized by four sets of motion parameters from [3], shown in Table 4.1. After synthesis, the MVs are corrupted by independent zero-mean Gaussian noise (i.e. Type-1 outliers) with standard deviation $\sigma \in \{0.7, 1.5, 2.2, 3.0\}$, in both *X*- and *Y*- components, and by groups of connected MVs pointing in a random direction (i.e. Type-2 outliers) simulating the foreground moving region. We evaluate the performance of the proposed algorithm in two scenarios: a single moving object and multiple moving objects. In the case of a single moving object (the square in Fig. 4.4(a)), MVs of all the blocks in this object are set to $(-5, -5)$. This makes the square move diagonally across the frame, as shown in Fig. 4.4(a). In the case of multiple moving objects, three objects are simulated — square, circle, and triangle — and are shown in Fig. 4.4(b). MVs of their blocks are set to $(-5, -5)$, $(-6, -4)$ and $(2, -7)$, respectively. The sizes of these objects correspond to 6%, 10% and 18 % of the overall MV field size.

Table 4.1: Global motion parameters used for testing

Model	Motion parameters
GM 1	$\mathbf{m}$=[0.9, 0, 10.4238, 0, 0.95, 5.7927, 0, 0]
GM 2	$\mathbf{m}$=[0.9964, –0.0249, 1.0981, 0.0856, 0.9457, –7.2, 0, 0]
GM 3	$\mathbf{m}$=[0.9964, –0.0249, 6.0981, 0.0249, 0.9964, 2.5109, -2.7×10^{-5}, 1.9×10^{-5}]
GM 4	$\mathbf{m}$=[1, 0, 4.4154, 0, 1, 0, -1.13×10^{-5}, 0]



(a)



(b)

Figure 4.4: (a): Scenario-4 with GM 1: synthetic MV field (16×16 MBs) corrupted by both Gaussian noise ($\sigma = 2.2$) and a foreground moving region of size 6×6 MVs; (b): Scenario-5 with GM 1, synthetic MV field (8×8 MBs) with three foreground moving regions.

We consider five test scenarios shown in Table 4.2. Scenarios 1 through 4 are for a single moving object, with the object size ranging from 2% up to 20% and with different noise variances. Scenario 5 involves multiple moving objects, with the total size of the three moving objects summing up to 34% of the MV field. In these test scenarios, global motion is the dominant motion in the scene, i.e. the background occupies more than 50% of the overall MV field size. If the background were to occupy less than 50% of the MV field size our method (as well as most other GME methods) would have trouble estimating GM parameters, but in these cases it would be hard to justify calling background motion "global motion."

Table 4.2: Test scenarios

Test Scenarios	MVF Block Size	Gaussian Noise (σ) (Type-1 outliers)	Moving region percentage (Type-2 outliers)
Scenario 1	16×16	0.7	Square: 9×9 (20 %)
Scenario 2	16×16	3.0	Square: 3×3 (2 %)
Scenario 3	16×16	1.5	Square: 6×6 (10 %)
Scenario 4	16×16	2.2	Square: 6×6 (10 %)
Scenario 5	8×8	2.2	Circular (18 %), Square (11 %), Triangle (5 %)

Figure 4.5 shows the outliers identified by the proposed method in scenarios 1, 2, 4 and 5 (top to bottom) on GM 4 at the end of the 2nd, 4th and 6th iteration (left to right). Type-1 outliers are shown in blue, Type-2 outliers are shown in green, while those that are identified as both Type-1 and Type-2 are shown in black. LS regression is used for GME in each iteration. Results for scenario 3 follow a similar trend as in scenario 4. As can be seen in the figure, Type-1 outliers remain the same through the iterations since their detection is based on local similarity, while Type-2 outliers may change with each iteration. We can observe that the accuracy of Type-2 outlier detection improves as the number of iterations increases. This is mainly due to the improvement of the accuracy of the estimated GM parameters, which in turn benefits motion segmentation.

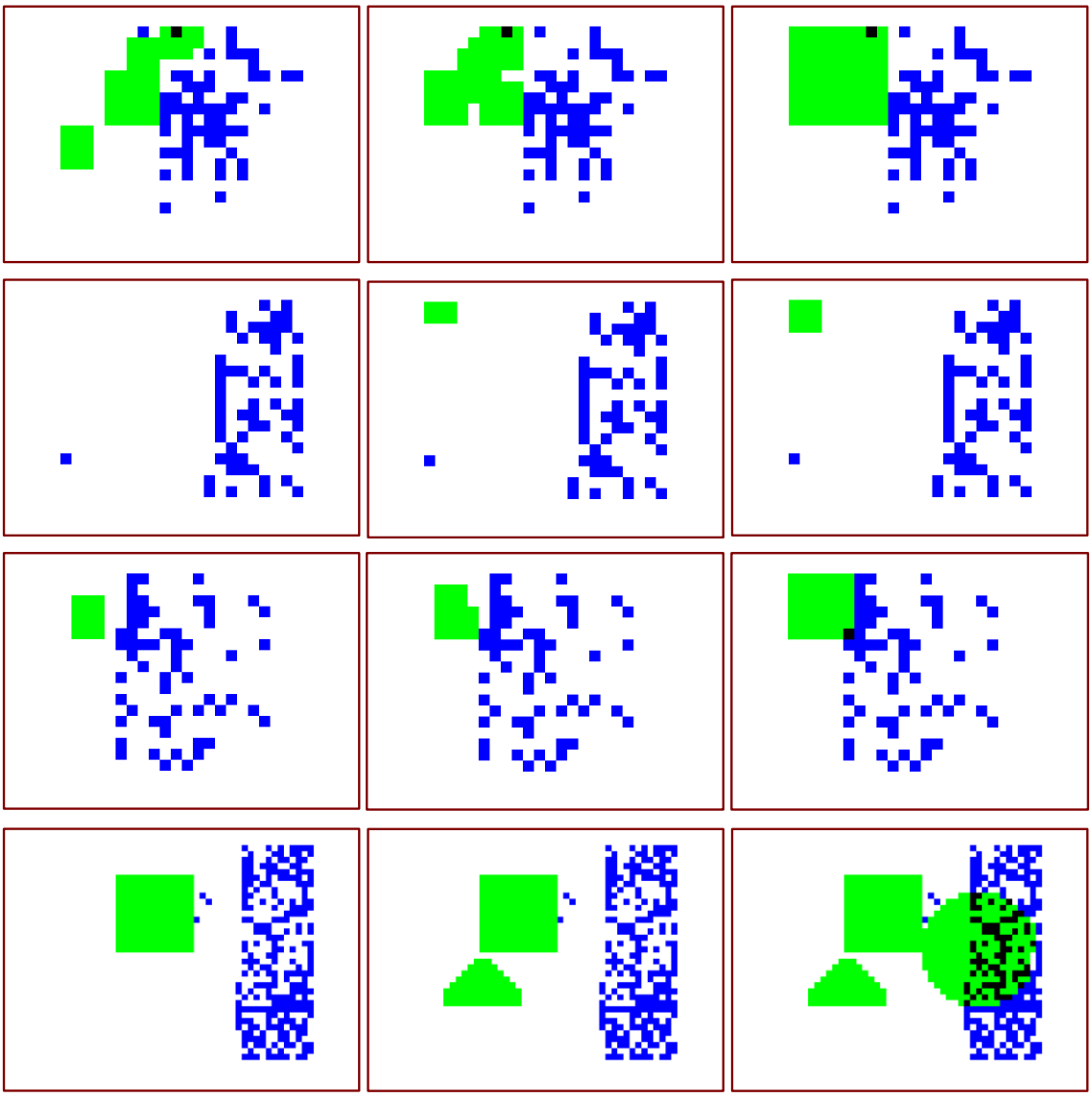


Figure 4.5 Identified MV outliers by proposed method. [Top to bottom]: scenario 1, 2, 4 and 5, [Left to right]: the 2nd, 4th, and 6th iterations. Type-1 and Type-2 outliers in the first frame of the MV field synthesized by GM 4 for scenario 1, 2 and 4. Type-1 outliers are shown in blue, Type-2 in green, while black blocks indicate MVs identified as both Type-1 and Type-2 outliers.

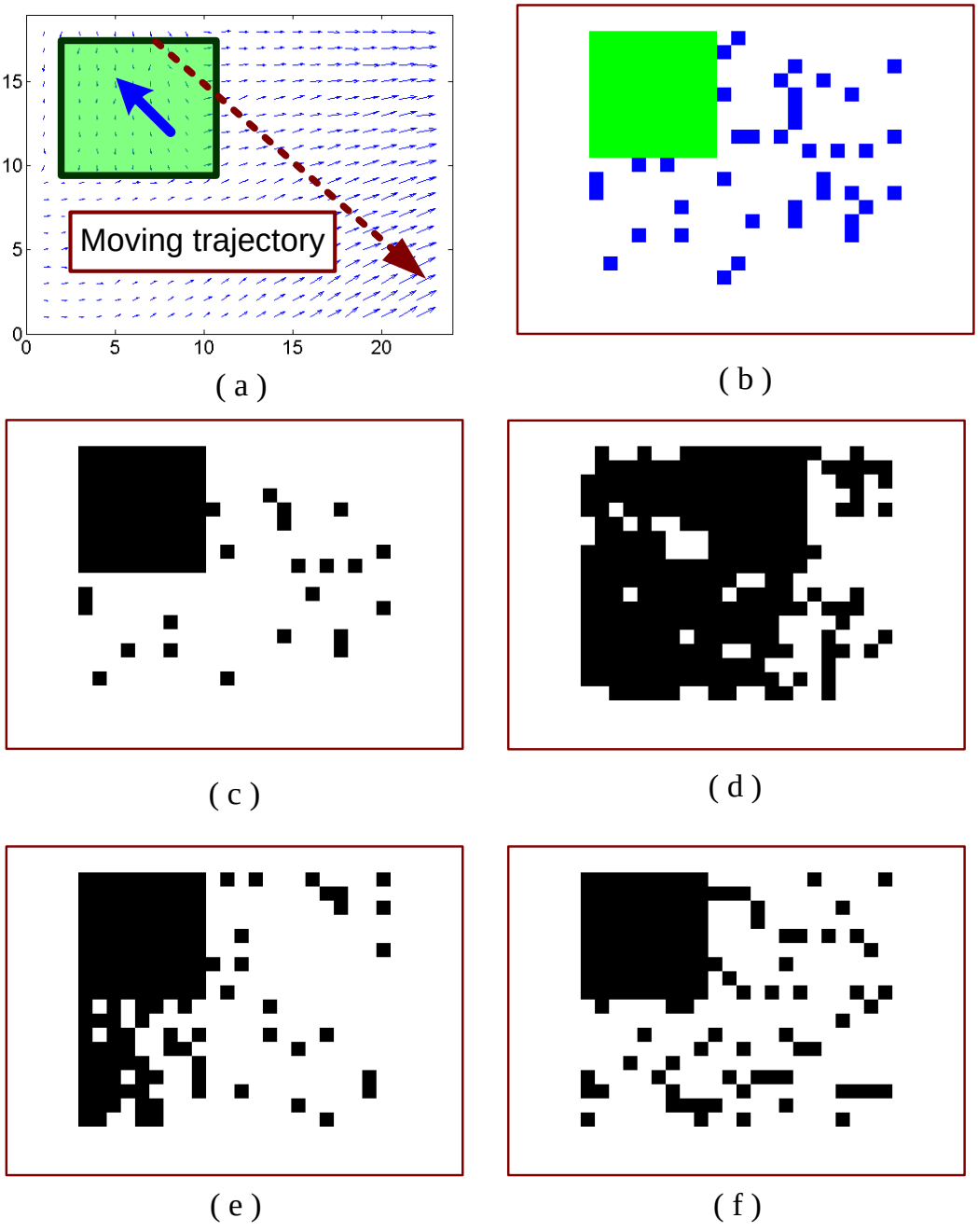


Figure 4.6: MV outlier detection from synthetic MV field using GM-4 with the setting of scenario 1, (a) the synthetic MV field, (b) the proposed approach, (c) LSS-ME, (d) filter from [23], (e) RANSAC, and (f) GD.

Figure 4.6 shows the final MV outlier maps obtained from the MV field synthesized using GM 4 in scenario 1. The MV field itself is shown in Fig. 4.6(a). Other subfigures show the results of the proposed method (Fig. 4.6(b)), LSS-ME (Fig. 4.6(c)), the MV outlier removal filter from [23] (Fig. 4.6(d)), RANSAC (Fig. 4.6(e)) and GD (Fig. 4.6(f)). All results are from the 20^{th} iteration of the corresponding algorithm, except for RANSAC, which we let run for up to 500 iterations. We see that the proposed method generates a similar outlier map as LSS-ME; both of these methods, as well as GD and RANSAC, correctly identify outliers associated with the moving object, but only our method is able to distinguish Type-1 and Type-2 outliers. Meanwhile, the outlier removal filter from [23] generally removes too many MVs due to its hard-thresholding strategy.

As a measure of GME accuracy, we use the signal-to-noise ratio (SNR) between the MV field generated by the GM parameters in Table 4.1 and the MV field generated by the estimated parameters, as in [3]. Figure 4.7 shows the average SNR verses the number of iterations for several GME methods on an MV field synthesized using GM 4 in scenario 1 (two subfigures on the top), scenario 3 (two subfigures in the middle) and scenario 5 (two subfigures at the bottom). The left subfigures correspond to the case where gradient descent is used for regression. The methods in this category are the proposed method incorporating segmentation (SEG_GD), iterative robust gradient descent estimator (GD from [3]), the MV outlier removal filter from [23] coupled with GD regression (FLT_GD) and RANSAC coupled with GD regression (RAN_GD) [6] [22]. The right subfigures correspond to least-squares regression and include the proposed method (SEG_LS), LSS-ME [11], the MV outlier removal filter from [23] coupled with LS regression (FLT_LS) and RANSAC coupled with LS regression (RAN_LS) [6] [22].

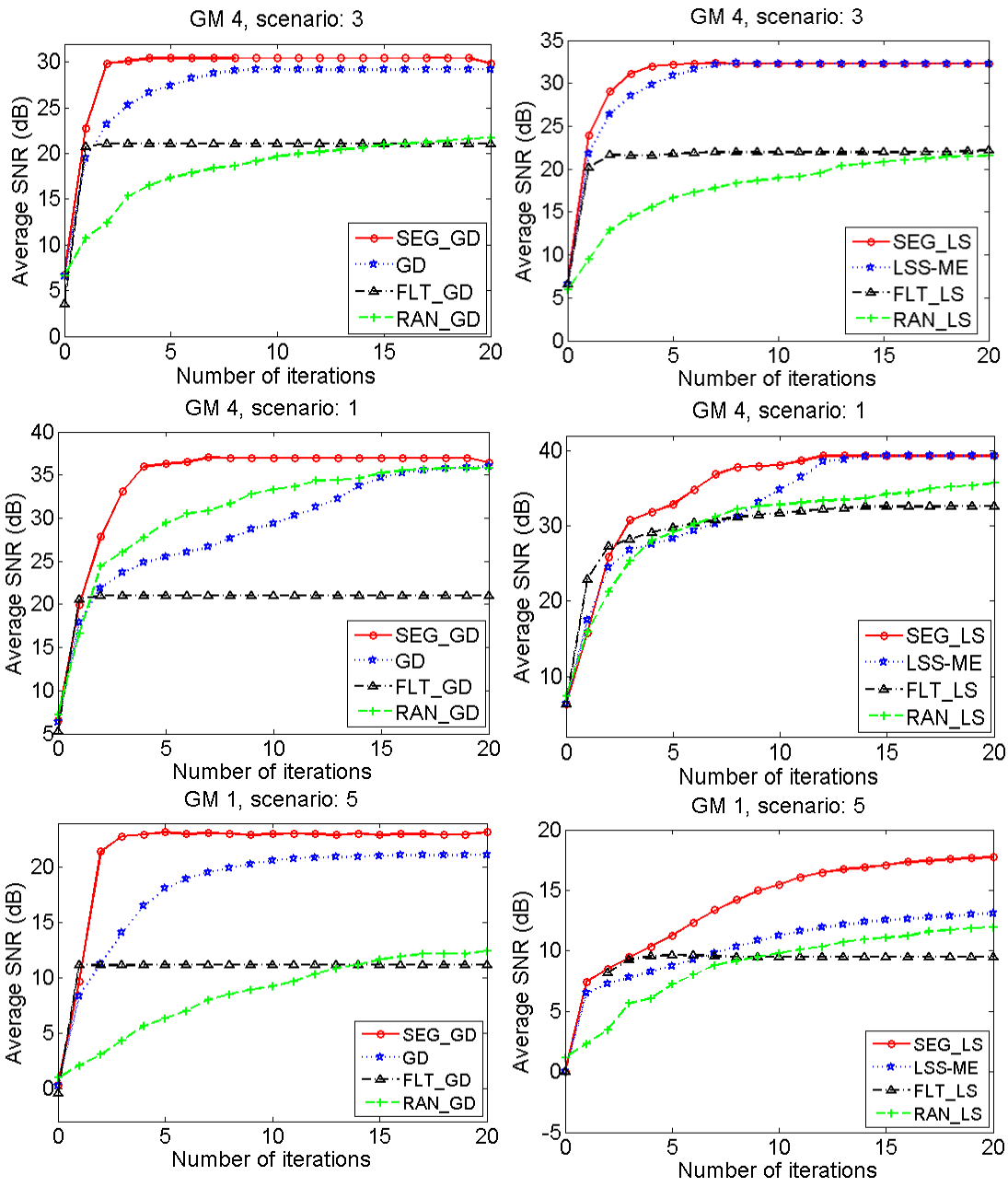


Figure 4.7: SNR vs. the number of iterations with synthetic MV fields corrupted by both zero-mean Gaussian noise (Type-1 outliers) and moving regions (Type-2 outliers) in scenarios 3, 1, and 5 (top to bottom). [Left]: GD regression, [Right]: LS regression.

Comparing left and right subfigures we see that LS regression provides more accurate GME than GD regression when the percentage of Type-2 outliers is relatively

low, while GD regression seems to perform better in the bottom two subfigures with outlier percentage of 34% (scenario 5 with three moving objects). To explain this observation, let S be the set of MVs upon which regression is performed and S^* be the set of background MVs. Ideally, regression should be performed on S^*, except that S^* is not known in advance. Finding S^* is the purpose of outlier removal. GD regression runs the risk of getting stuck in a local error minimum on S, while LS finds, in one step, the global error minimum on S. When the outlier percentage is small S does not contain too many outliers (i.e. S is close to S^*) even in the early iterations of outlier rejection (i.e. segmentation), which favours LS. However, as the outlier percentage increases the local minima on S achieved by GD may prove to be closer to the global minimum on S^* compared to the global minimum on S achieved by LS. We believe that this is the reason why GD outperforms LS in the bottom subfigures in Fig. 4.7. Note, however, that the SNR achieved by both methods in this case is much lower (10-25 dB) than what they were able to achieve under more favourable conditions in the top and middle subfigures.

The outlier removal filter from [23] converges with the fewest iterations, but with the least accurate parameter estimates. Our outlier removal involving segmentation converges with the second lowest number of iterations and gives the most accurate estimates with both regression techniques. Additionally, our method produces a coarse segmentation map of moving regions in the scene, while other methods do not provide this feature. As in [28], we find that RANSAC needs the largest number of iterations for convergence and seems to work better in the presence of Type-2 outliers with low noise (the two subfigures in the middle of Fig. 4.7) than in the cases where the noise level is high (the four subfigures at the top and bottom of Fig. 4.7). The results with other GM models follow similar trends.

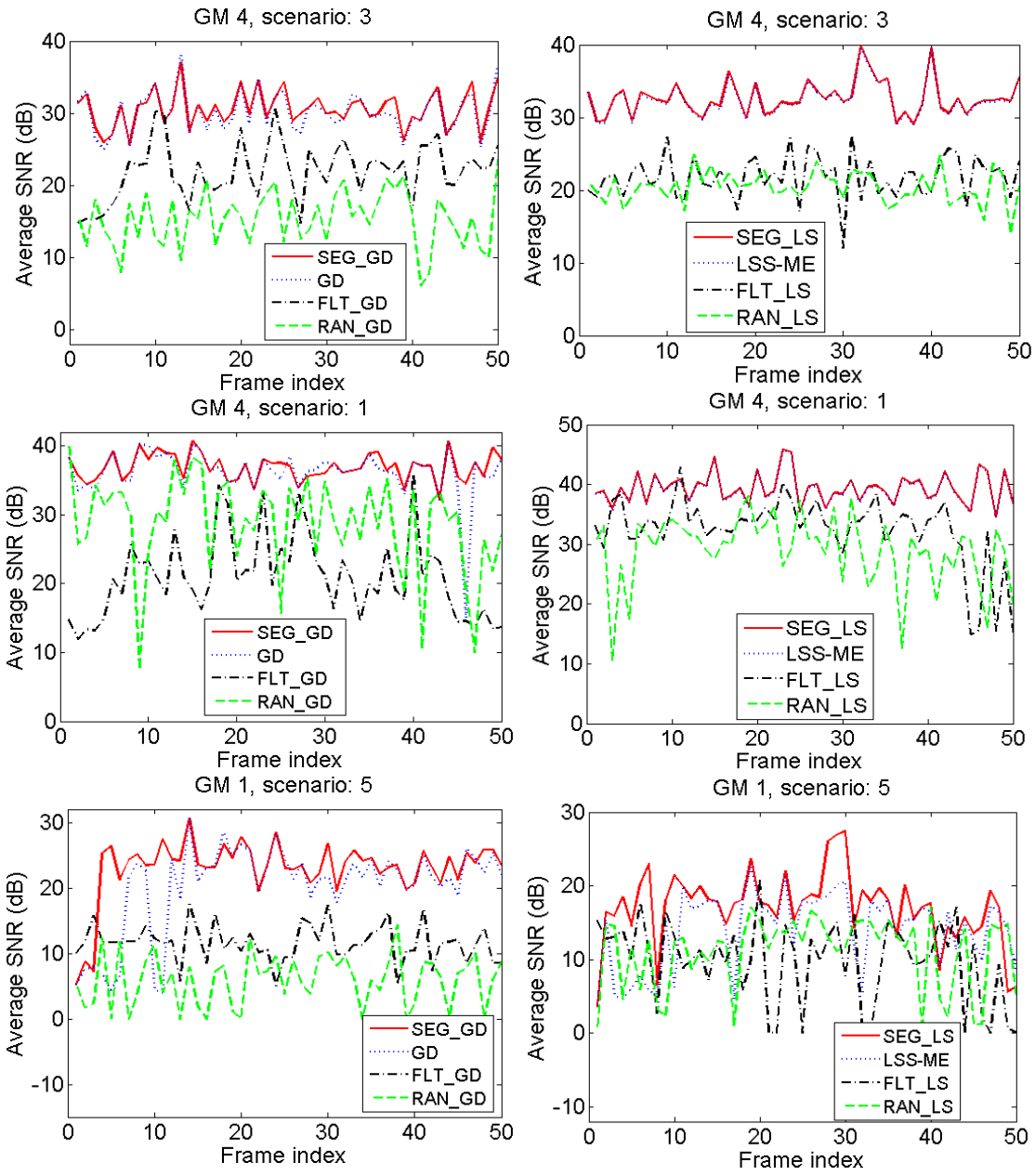


Figure 4.8: Converged SNR per frame on synthetic MV fields corrupted by both zero-mean Gaussian noise (Type-1 outliers) and moving regions (Type-2 outliers) in scenarios 3, 1, and 5 (top to bottom). [Left]: GD regression, [Right]: LS regression.

Table 4.3: SNR on synthetic MV field (dB), GME using gradient descent regression

GM Model	GME Algorithms	Test Scenarios				
		1	2	3	4	5
GM 1	**SEG_GD**	**32.55**	**20.18**	**26.80**	**23.09**	**20.59**
	GD	31.87	18.69	26.27	21.92	19.89
	FLT_GD	12.93	7.64	14.45	11.29	8.56
	RAN_GD	21.55	4.40	12.08	6.48	4.50
GM 2	**SEG_GD**	**35.71**	**23.47**	**30.73**	**25.89**	**22.65**
	GD	34.60	23.38	29.77	25.66	21.77
	FLT_GD	16.77	12.03	21.39	16.65	12.32
	RAN_GD	26.46	8.75	15.75	11.40	4.29
GM 3	**SEG_GD**	**34.50**	**21.10**	**27.84**	**23.94**	**20.86**
	GD	34.03	20.43	27.14	23.30	20.35
	FLT_GD	17.98	9.76	17.46	12.68	8.77
	RAN_GD	21.78	5.13	12.35	8.20	3.03
GM 4	**SEG_GD**	**37.05**	**24.16**	**30.80**	**27.12**	**20.28**
	GD	35.73	23.44	30.19	25.52	19.88
	FLT_GD	21.98	13.17	21.83	17.01	10.54
	RAN_GD	26.46	8.26	16.07	10.68	4.95

Figure 4.8 shows the "converged" SNR per frame for the same GME methods as in Fig. 4.7. This "converged" SNR corresponds to 20 GME iterations for all methods except those involving RANSAC, which we let run for up to 500 iterations. From the figure, we can observe that with GD regression, the proposed SEG_GD and GD from [3] perform better than other methods with a reasonably large margin, while SEG_GD has a slightly higher SNR than GD. With LS regression, the proposed SEG_LS and LSS-ME from [11] achieve very close SNRs and both outperform other methods when there is a single moving object in the scene (top and middle subfigures). In these two cases, SEG_LS and LSS-ME provide virtually the same SNR. Note that both methods employ the same LS regression; hence, the only difference is possibly in the set of outliers rejected by the two methods. With a single object in the scene (top and middle subfigures in Fig. 4.8), the set of outliers appears to be very similar, if not the same, for both methods, so the SNR is virtually identical. However, when the number of objects

increases to three and the Type-2 outlier percentage increases to 34% (bottom subfigure in Fig. 4.8) SEG_LS clearly outperforms LSS-ME.

Table 4.4: SNR on synthetic MV field (dB), GME using least squares regression

GM Model	GME Algorithms	Test Scenarios				
		1	2	3	4	5
GM 1	**SEG_LS**	**34.83**	**20.32**	**28.74**	**23.77**	**18.25**
	LSS-ME	34.83	20.31	28.72	23.55	17.46
	FLT_LS	21.57	6.77	12.39	8.10	4.17
	RAN_LS	24.65	9.28	16.54	12.61	6.64
GM 2	**SEG_LS**	**38.41**	**23.89**	**31.41**	**27.61**	**21.48**
	LSS-ME	38.40	23.88	31.39	27.41	20.61
	FLT_LS	31.19	11.72	21.64	14.82	11.66
	RAN_LS	28.13	12.51	20.35	16.58	10.30
GM 3	**SEG_LS**	**36.00**	**21.53**	**29.61**	**25.17**	**20.52**
	LSS-ME	36.00	21.51	29.60	24.81	19.74
	FLT_LS	26.88	7.05	16.80	11.01	8.65
	RAN_LS	26.15	10.76	17.67	13.71	8.38
GM 4	**SEG_LS**	**39.29**	**24.57**	**32.78**	**28.55**	**15.70**
	LSS-ME	39.29	24.58	32.70	28.13	15.07
	FLT_LS	31.74	10.71	21.56	16.67	12.52
	RAN_LS	29.68	13.17	21.77	16.28	7.12

The converged SNR values averaged over all frames for all four GM models are listed in Tables 4.3 and 4.4, for GD and LS regression, respectively. In Table 4.3, we observe that the proposed scheme (SEG_GD) has the overall best performance. In Table 4.4, the proposed SEG_LS achieves a very similar average SNR as LSS-ME from [11] in test scenarios 1–3 and has a slight advantage (0.2–0.7 dB) in scenarios 4 and 5. Both SEG_LS and LSS-ME have a relatively large advantage over the other two methods.

Instead of computing the SNR between the estimated motion field and the true field, as was done above and in [3], we could also compute the estimation error directly

on the eight motion parameters. In Fig. 4.9, we show the squared L_2 norm of the parameter error, computed as:

$$E_{\mathbf{m}} = \|\mathbf{m} - \hat{\mathbf{m}}\|_2^2 = \sum_{i=0}^{7} (\mathbf{m}_i - \hat{\mathbf{m}}_i)^2 \qquad \text{Equation 4.6}$$

Figure 4.9 (the left sub-image) corresponds to an MV field synthesized by GM-1 parameters and corrupted with 10% Type-2 outliers and noise with $\sigma \in [0.5,3.0]$. Figure 4.9 (the right sub-image) corresponds to an MV field synthesized by GM-4 and corrupted by noise with $\sigma = 0.7$ and a Type-2 outlier percentage between 5% and 25%. In both cases, we see that SEG_GD provides a better estimate of the true parameters than other methods, as expected based on the previously presented SNR results.

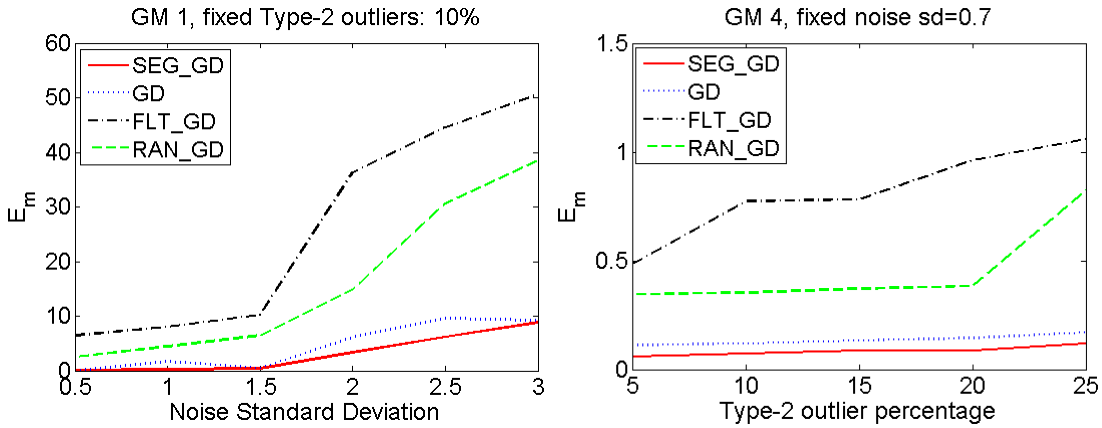


Figure 4.9: Estimation error $E_{\mathbf{m}}$ on the 8 motion parameters. [Left]: GM-1 field corrupted by 10% Type-2 outliers and noise with $\sigma \in [0.5, 3.0]$, [Right]: GM-4 field corrupted by noise with $\sigma = 0.7$ and various Type-2 outlier percentages.

4.4.2. Evaluation on Real Video Sequence

In this section, we test the proposed method on real video sequences. Seven sequences are used in the experiments — *Flower Garden*, *Stefan*, *Tempete*, *City*, *Coastguard*, *Waterfall*, and *Mobile Calendar* — all CIF resolutions with a frame rate at 30 frames per second. These sequences contain a significant amount of camera motion.

The evaluation methodology adopted in this work is to use GME to estimate the GM model parameters and then perform global motion compensation by warping the target frame onto the reference frame according to the model using bilinear interpolation [26]. If the sequence contains only camera motion, and if GME is accurate, we should expect the frames compensated by global motion to be very close to the original frames. The similarity can be measured using the conventional PSNR. The evaluations are conducted on two kinds of MV fields: one generated by exhaustive search motion estimation (8×8 blocks), and the other by Rate-Distortion Optimized (RDO) H.264 motion estimation using the NAL-SIM simulator [32] (IPPP GOP structure, at 512 kbps, with the smallest MV block size of 8×8). The MV field from H.264 RDO motion estimation is then normalized by block splitting to obtain a uniformly sampled MV field on 8×8 blocks.

We first evaluate MV outlier removal on *Flower Garden*. This sequence includes the tree trunk as the foreground static object. The background here consists of objects far from the camera — houses, garden and the sky. The tree trunk is too close to the camera to fit into the 8-parameter motion model of the background. Hence, even though the tree trunk is not really moving relative to the background, it will have apparent motion that is distinct from the background. It will therefore represent a source of Type-2 outliers, as discussed in the definition of Type-2 outliers in Section 2.2.2. In Fig. 4.10, we show the final MV outlier maps obtained from the MV field in Fig. 4.10(a), which corresponds to frame #3 of *Flower Garden*. The results correspond to several outlier detection methods: the proposed method shown in Fig. 4.10(b), LSS-ME in Fig. 4.10(c), the MV outlier removal filter [23] in Fig. 4.10(d), RANSAC in Fig. 4.10(e), and plain robust iterative GD in Fig. 4.10(f). We see that the proposed method finds Type-2 outliers (interior of a moving object — tree trunk, shown in green) and Type-1 outliers ("noisy" MVs near object boundaries, shown in blue), as well as those MVs which fit both Type-1 and Type-2 outlier definitions (shown in black). Meanwhile, LSS-ME only detects part of the moving object. The outlier map of filter [23] is only able to detect the Type-1 outliers. RANSAC and iterative GD generate similar outlier maps and manage to detect the moving object similar to our method, but they typically require a larger number of iterations to achieve this (in these experiments 100–200 iterations were required). Furthermore, only our method is able to distinguish Type-1 and Type-2 outliers, which can be useful for subsequent video processing tasks (e.g. pixel-based segmentation).

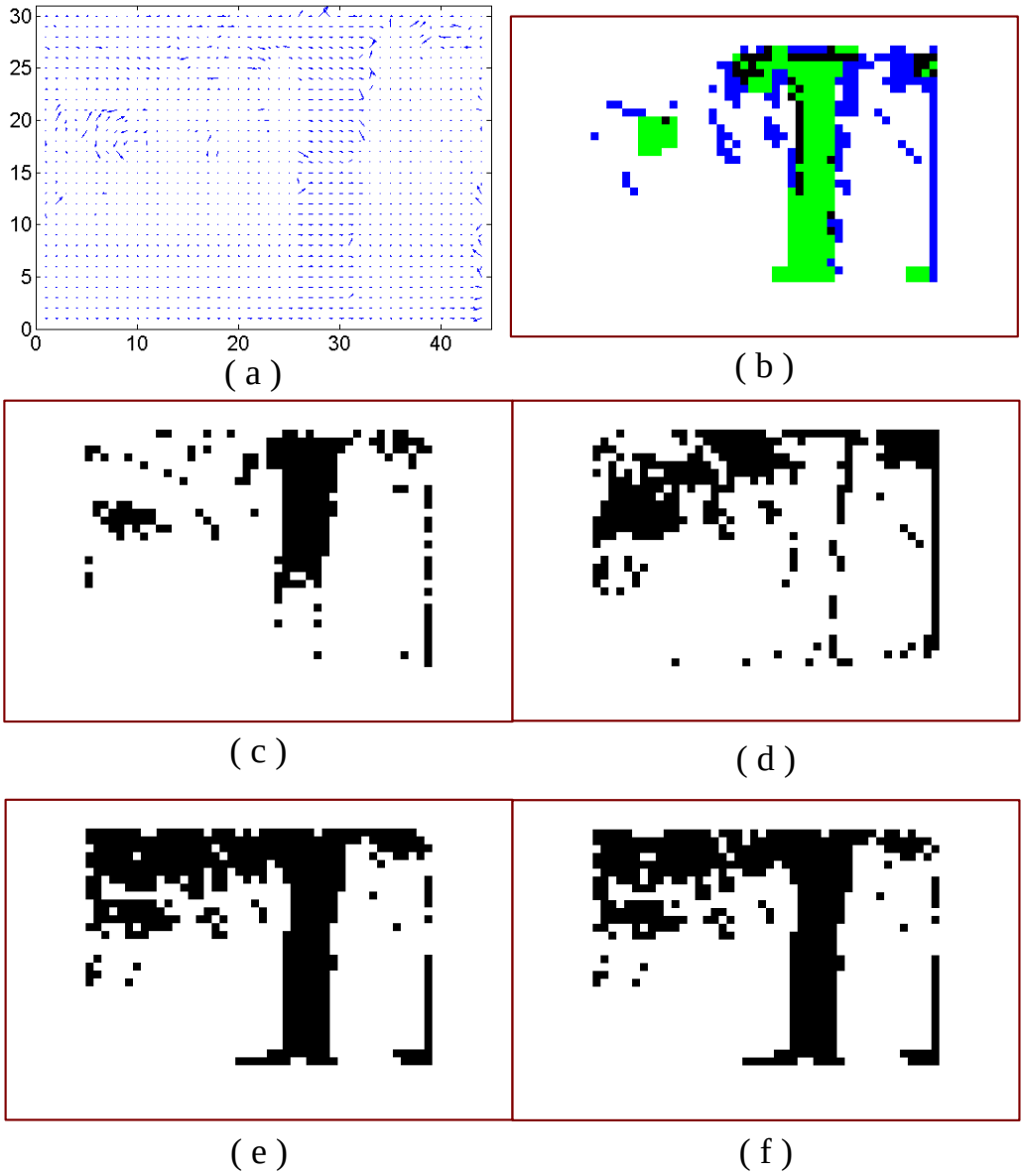


Figure 4.10: MV outlier detection from sequence *Flower Garden*, (a): original MV field from compressed video stream, (b): the proposed joint segmentation and GME approach, (c) LSS-ME, (d): MV outlier removal filter from [23], (e): RANSAC, and (f): plain iterative GME using gradient descent.

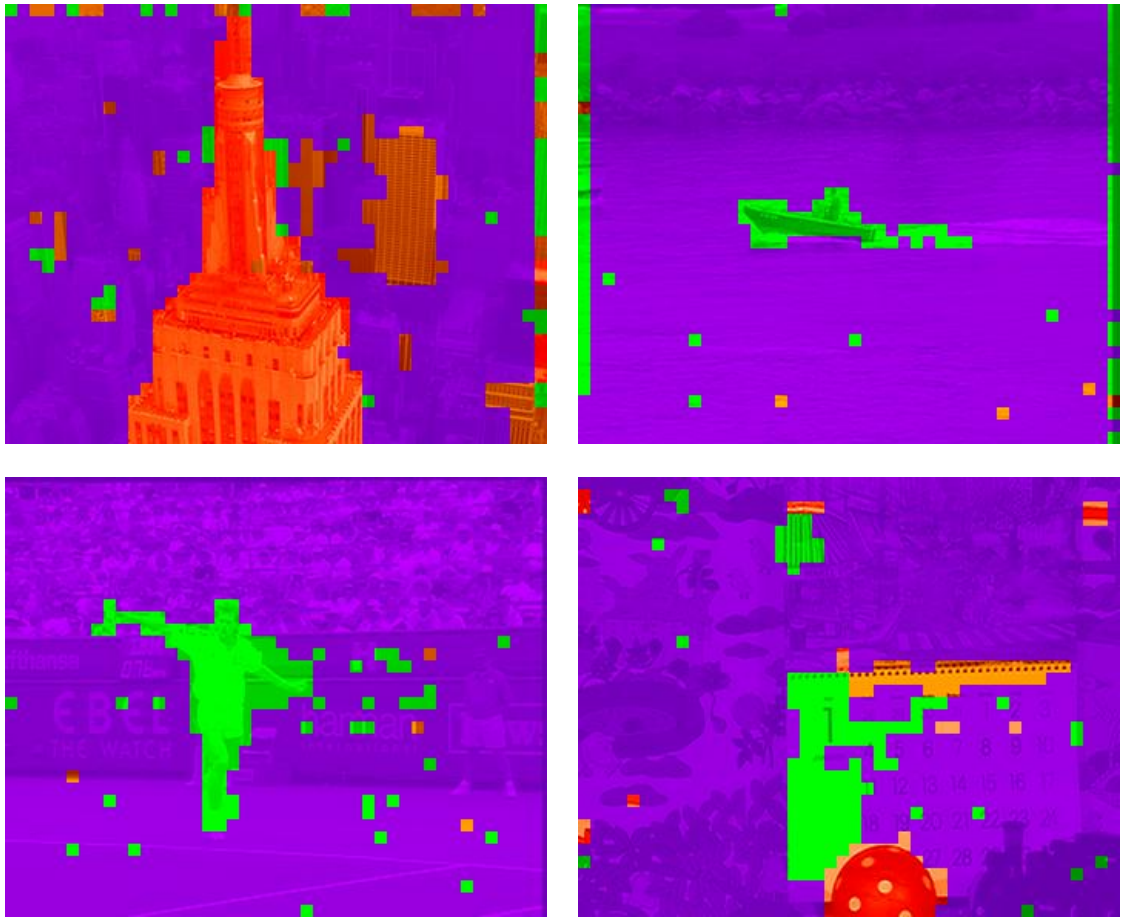


Figure 4.11: Bayesian segmentation of *City* (frame #3), *Coastguard* (frame #3), Stefan (frame #15), and *Mobile Calendar* (frame #10).

In Fig. 4.11, we show several segmentation results obtained by the proposed joint segmentation-GME on *City*, *Coastguard*, *Stefan* and *Mobile Calendar*. All results correspond to the end of the third iteration of joint segmentation and GME. Among these sequences *Coastguard*, *Stefan* and *Mobile Calendar* include objects that are actually moving relative to the background, while *City* contains a static foreground object (the Empire State building) which is too close to the camera compared to other buildings in the scene and therefore does not fit the 8-parameter motion model of the background.

Finally, we evaluate the performance of the proposed method and other GME methods on whole sequences. Since the GM characteristics do not change much between consecutive frames, we use the estimated GM parameters from the previous frame to initialize GME in the current frame. For all methods, we use the same termination conditions as in [3], i.e. if $\mathbf{m}^{(i-1)}$ and $\mathbf{m}^{(i)}$ are the GM parameters estimated at $(i-1)$ -th and i-th iteration, and $\Delta\mathbf{m} = \mathbf{m}^{(i)} - \mathbf{m}^{(i-1)}$, then the process terminates if Δm_2 and Δm_5 (translational components) are less than 10^{-3}, and other Δm's are less than 10^{-5}. Further, to balance accuracy and computational complexity, we set the maximum number of iterations to be 3 for the proposed joint GME and segmentation, 3 for the filter from [23], 6 for the iterative GD and LSS-ME and 500 for RANSAC.

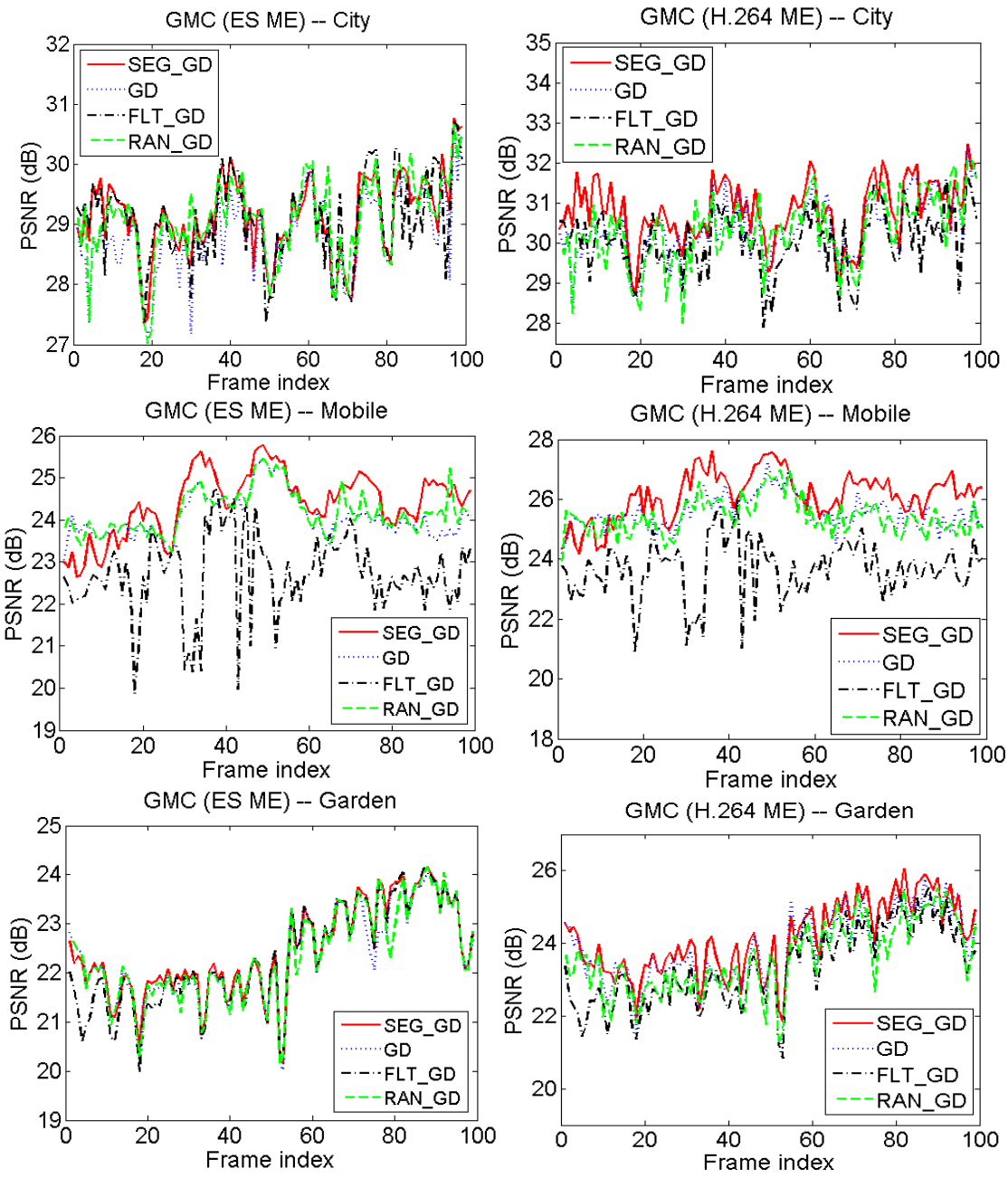


Figure 4.12: Global motion compensation (GMC) performance evaluation using gradient descent regression: [Left]: MVs generated by exhaustive search, [Right]: MVs generated by RDO H.264 motion estimation.

Table 4.5: GME using GD regression GMC performance with ES ME, PSNR in dB

Sequences	SEG_GD	FLT_GD	GD	RAN_GD
Flower Garden	22.43	22.30	22.35	22.33
Tempete	27.79	24.98	26.53	26.67
Stephan	24.72	22.17	24.56	24.80
City	29.41	29.07	28.82	29.16
Coastguard	26.77	26.63	26.60	26.62
Waterfall	34.98	24.36	34.73	34.75
Mobile Calendar	24.54	22.69	24.07	24.13
Average	**27.24**	**24.59**	**26.80**	**26.92**

GMC PERFORMANCE WITH H.264 ME, PSNR IN dB

Sequences	SEG_GD	FLT_GD	GD	RAN_GD
Flower Garden	22.47	22.29	22.41	22.37
Tempete	28.28	25.51	26.97	26.81
Stephan	25.03	21.91	24.70	25.30
City	28.77	28.72	28.68	29.01
Coastguard	27.31	26.71	26.37	26.93
Waterfall	35.48	24.55	34.87	35.27
Mobile Calendar	24.74	22.73	24.27	24.35
Average	**27.45**	**24.63**	**26.90**	**27.15**

GME PROCESSING TIME IN MILLISECONDS

Sequences	SEG_GD	FLT_GD	GD	RAN_GD
Flower Garden	167.6	26.2	33.2	136.6
Tempete	167.7	19.2	38.3	109.4
Stefan	193.9	22.0	38.0	182.9
City	180.9	24.5	42.6	125.0
Coastguard	173.6	24.4	38.4	110.6
Waterfall	149.0	26.8	46.4	93.3
Mobile Calendar	179.2	22.5	46.9	116.2
Average	**173.1**	**23.7**	**40.5**	**124.9**

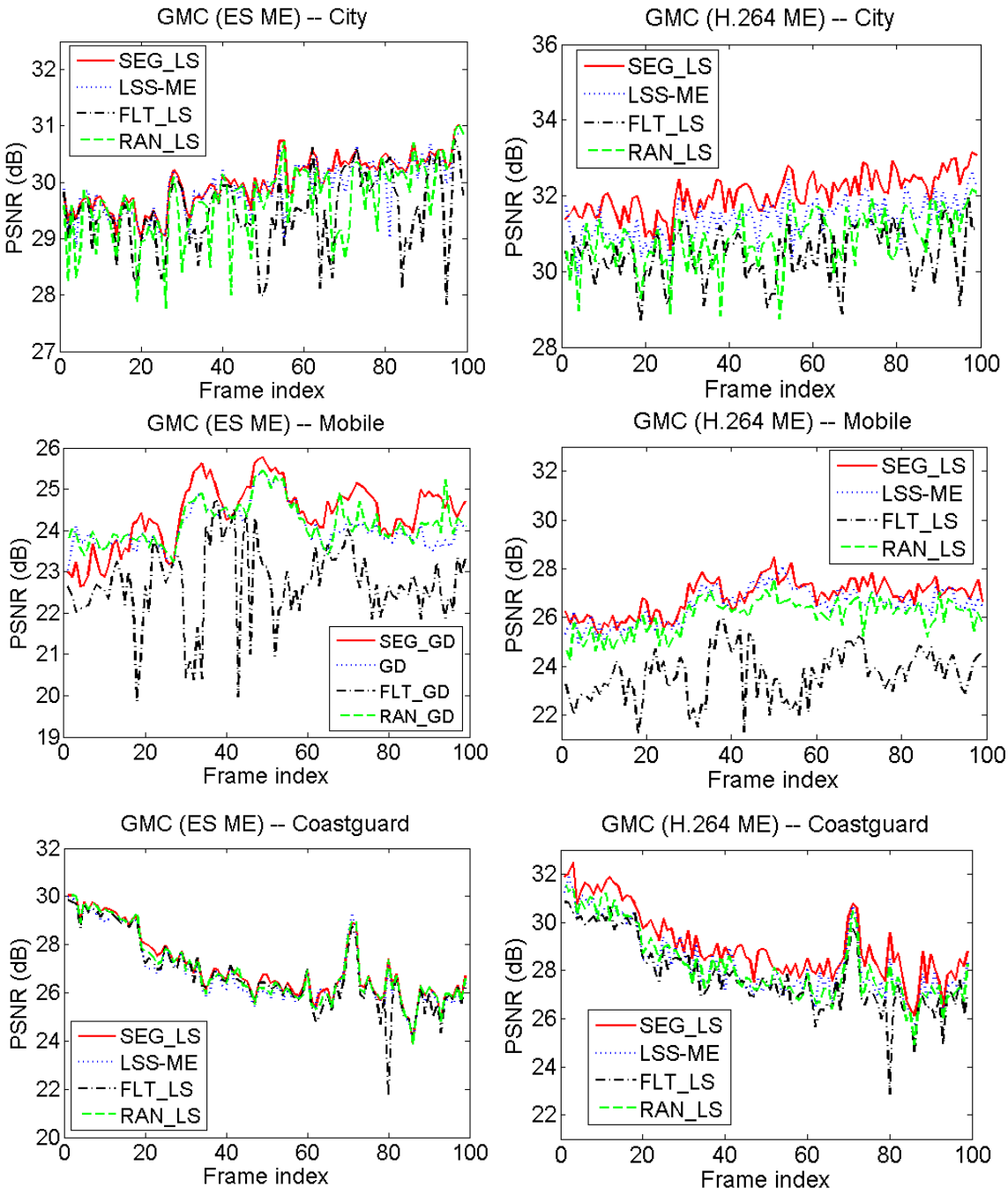


Figure 4.13: Global motion compensation (GMC) performance evaluation using least-squares regression: [Left]: MVs generated by exhaustive search, [Right]: MVs generated by RDO H.264 ME

Table 4.6: GME using LS regression GMC performance with ES ME, PSNR in dB

Sequences	SEG_LS	FLT_LS	LSS-ME	RAN_LS
Flower Garden	22.16	21.91	22.46	21.92
Tempete	27.90	24.71	27.66	27.83
Stefan	24.63	21.92	24.60	24.73
City	30.00	29.44	29.88	29.62
Coastguard	27.08	26.76	26.82	26.95
Waterfall	35.51	24.29	35.18	35.50
Mobile Calendar	24.86	22.66	24.97	24.75
Average	**27.45**	**24.53**	**27.38**	**27.33**

GMC PERFORMANCE WITH H.264 ME, PSNR IN dB

Sequences	SEG_LS	FLT_LS	LSS-ME	RAN_LS
Flower Garden	21.99	21.98	22.58	22.06
Tempete	28.46	24.90	27.53	27.67
Stefan	24.75	22.30	24.65	25.07
City	29.97	29.55	29.99	29.67
Coastguard	27.40	26.64	26.83	27.09
Waterfall	36.08	24.06	35.05	35.23
Mobile Calendar	25.06	23.01	25.11	24.81
Average	**27.62**	**24.63**	**27.39**	**27.41**

GME PROCESSING TIME IN MILLISECONDS

Sequences	SEG_LS	FLT_LS	LSS-ME	RAN_LS
Flower Garden	259.3	94.8	298.6	123.7
Tempete	272.4	33.9	442.0	94.8
Stefan	236.2	55.5	479.3	137.5
City	276.8	104.2	488.2	116.2
Coastguard	276.3	115.0	478.5	102.6
Waterfall	265.6	32.1	477.5	83.4
Mobile Calendar	274.5	76.0	464.6	111.2
Average	**265.9**	**73.1**	**446.9**	**109.9**

Again, we test both regression techniques — GD and LS — on MV fields generated by exhaustive search motion estimation (ES ME) and RDO H.264 motion estimation (H.264 ME). The results with GD regression are shown in Fig. 4.12 and Table 4.5, while the results with LS regression are shown in Fig. 4.13 and Table 4.6. The tables also show the average processing time per frame, measured in MATLAB on a standard desktop PC with an Intel Pentium CPU at 3.0 GHz and 2 GB of RAM. This processing time includes all filtering and GME operations. Note that motion estimation time is not included in the processing time in our experiments, since in practise, MVs will be read directly from the compressed video bitstream.

From Table 4.5, we can see SEG_GD has the overall best PSNR performance among the methods utilizing GD regression on both kinds of MV fields. By incorporating motion segmentation into GME, and using a maximum of 3 iterations, we can achieve a 0.4dB PSNR gain (ES ME) and a 0.55dB PSNR gain (H.264 ME) over plain GD. By using the same GD regression technique motion segmentation can better identify MV outliers and lead to a more accurate GM estimation.

With LS regression (Fig. 4.13 and Table 4.6), we again observe that our SEG_LS achieves the best PSNR performance among the four tested methods. Also, LS regression turns out to be slightly better than GD regression for GME with the 8-parameter perspective model. LSS-ME achieves similar PSNR as the proposed SEG_LS, about 0.2dB lower with H.264 ME and within 0.1dB with ES ME, which reinforces the findings obtained on synthetic MV fields where the two methods also achieved very close SNR.

Earlier results on synthetic MV fields indicate that MV noise spread throughout the MV field can be very detrimental to RANSAC because it makes it more difficult to find noise-free MVs. In real sequences, MV noise is usually found in the regions that lack texture or near the boundaries of moving objects. In the test sequences used in our experiments it appears that the number of such noisy MVs was relatively small and that outliers were mostly Type-2 outliers. In these sequences, large portions of the background contain sufficient texture to produce relatively accurate MVs, which gives RANSAC a good chance to select some of these MVs as inliers. Hence, RANSAC performed rather well in these tests with both GD and LS regression.

The price paid for improved accuracy of our methods is the processing time. With GD regression, our SEG_GD has the largest processing time per frame of 173 ms, which is not too high considering that the implementation is in MATLAB. With LS regression, our SEG_LS has the second largest processing time per frame, after LSS-ME. The reason why our methods have relatively large processing time despite requiring relatively few iterations is the fact that each iteration of our GME loop involves segmentation, which is costly. However, we believe that improved accuracy and the fact that a fairly good segmentation map is produced along the way justifies the increased processing time.

Furthermore, LS regression is more computationally demanding than GD regression in all cases, except when coupled with RANSAC outlier rejection. The complexity of LS regression is determined by the size of the MV field and the number of GM parameters. For example, for CIF resolution with 8×8 block size and 8-parameter perspective model, the $\mathbf{A}$ matrix has the dimensions 3168×8, so computing the least squares solution may take a relatively long time. However, when RANSAC is used, early iterations of RANSAC might not produce a consistent set of inliers, so GD regression may need a few iterations of its own (up to 5 in our experiments) to estimate global motion parameters, while LS regression always performs this estimation in one step. This is the reason why RAN_GD takes slightly more time than RAN_LS.

Finally, we compare our proposed algorithm with MPEG-4 GME [4]. The MPEG-4 GME is a 3-level hierarchical pixel-based approach, in which the outlier pixels are removed by thresholding the global motion compensation error. The qualitative advantage of our approach is that it distinguishes foreground objects through motion segmentation, whereas MPEG-4 GME does not. The results of quantitative comparison are presented in Table 4.7. We report the GMC PSNR gain of SEG_LS and SEG_GD, and the processing time difference on both the ES and H.264 MV field. As the results show, our methods are more accurate than MPEG-4 GME on some sequences, less accurate on others, and have a slight advantage in accuracy when considering the average GMC PSNR over all test sequences used in our work. But the main advantage of our methods is speed, which is two orders of magnitude higher than that of MPEG-4 GME. It should be mentioned that our methods, as well as MPEG-4 GME, are implemented in MATLAB (and can all be further optimized), but the observed speed

advantage is indicative of the amount of data that the two types of methods have to deal with. MV-based methods usually need to process only two numbers (*X*- and *Y*-component of the MV) per block, while pixel-based methods have to deal with all pixel values in the block. Indeed, the reduction of complexity has often been cited as the key advantage of compressed-domain approaches compared to the pixel-based ones.

Table 4.7: Comparison with MPEG-4 GME GMC performance comparison, PSNR in dB

Sequences	MPEG-4 GME	Gain (ES ME)		Gain (H.264 ME)	
		SEG_GD	SEG_LS	SEG_GD	SEG_LS
Flower Garden	22.01	+0.42	+0.15	+0.46	−0.02
Tempete	28.45	−0.66	−0.55	−0.17	+0.01
Stefan	24.46	+0.26	+0.17	+0.57	+0.29
City	26.57	+2.84	+3.43	+2.20	+3.40
Coastguard	26.41	+0.36	+0.67	+0.90	+0.99
Waterfall	36.92	−1.94	−1.41	−1.44	−0.84
Mobile Calendar	25.11	−0.57	−0.25	−0.37	−0.05
Average	27.13	+0.11	+0.32	+0.32	+0.49

GME PROCESSING TIME IN MILLISECONDS

Sequences	MPEG-4 GME	SEG_GD	SEG_LS
Flower Garden	55840	167.6	259.3
Tempete	32692	167.7	272.4
Stefan	30568	193.9	236.2
City	32608	180.9	276.8
Coastguard	35887	173.6	276.3
Waterfall	32229	149.0	265.6
Mobile Calendar	36182	179.2	274.5
Average	36572	173.1	265.9

5. Frame Composition

Frame composition fulfils the function of synthesizing a frame at a desired presentation time based on motion and object information that can be obtained using the methods from Chapters 2 - 4. In order to composite a frame, the major object features need to be obtained beforehand. These features include contour, texture, motion, depth order, and camera movements. In this work, we proposed and evaluated two object-based and one pixle-based frame synthesis methods: 1) Object-based synthesis using boundary match algorithm (BMA), 2) Object-based synthesis using ordinal depth, and 3) Pixel-based synthesis using the Linear Minimum Mean Square Error (LMMSE) criterion.

5.1. Object-Based Synthesis Using Boundary Match Algorithm

This approach was first proposed in our work [52] [93], and developed based on moving region segmentation. The main idea is to conduct a motion prediction on segmented moving regions, and move all blocks of the last received frame onto the future frame along the predicted moving trajectory to form a future frame.

5.1.1. Region-Based Motion Prediction

Motion segmentation identifies different moving regions in the previous frame [93]. The critical step towards frame synthesis is to predict how these moving regions will move onto the next frame. Since each region is a collection of blocks (8×8 blocks in our case), we could, for example, predict the motion of each block separately. However, we found that better results are obtained by assigning the same MV (in particular, MV centroid of the region) to all blocks in a particular region. This is not surprising, because regions are segmented in the first place as the collections of blocks that move in a coherent fashion. Although this approach improves the consistency of the predicted

motion of moving regions, it still suffers from the problems inherited from the block translation motion model used in conventional video coding [89], [94]. A single MV assigned to an entire moving region is obviously not appropriate in cases that involve camera rotation and zooming, and neither is the underlying block translation model. Nevertheless, this approach has proved to be better than assigning a separate MV to each block on standard test sequences used in our experiments. We speculate that the main reason is the modelling and estimation inaccuracies present in the MVs in the compressed bitstream, which become more pronounced when prediction is performed. A single MV for an entire moving region at least keeps that region intact over the predicted moving trajectory.

While providing a reasonably good prediction of the general direction of motion of a moving region, this approach may lead to blocking artifacts close to region boundaries. Therefore, we use a somewhat different strategy for the blocks close to region boundaries [95], [96], which we call variable-block-size motion prediction. Within each region, each block is classified as either an *internal block* or a *boundary block* in terms of its position and the energy of its prediction residual in Eq. 5.1, where $r(x, y)$ denotes the inter-frame prediction residual at position (x, y) within the 8×8 block and can be obtained through IDCT from bitstream.

$$E_{res} = \sum_{x=1}^{8} \sum_{y=1}^{8} \big(r(x, y)\big)^2 \qquad \text{Equation 5.1}$$

In this section, a block is classified as a *boundary block* if it has neighbours from more than one region, and its $E_{res} > 256$. The threshold 256 was determined experimentally and corresponds to an average prediction residual energy of 4 per pixel. Otherwise, if $E_{res} \leq 256$, the block is classified as an *internal block*, to which the region's centroid MV is assigned as explained above. Fig. 5.1 (a) and (b) show a sample frame with block-based boundary after motion segmentation, and a frame after boundary blocks are identified.

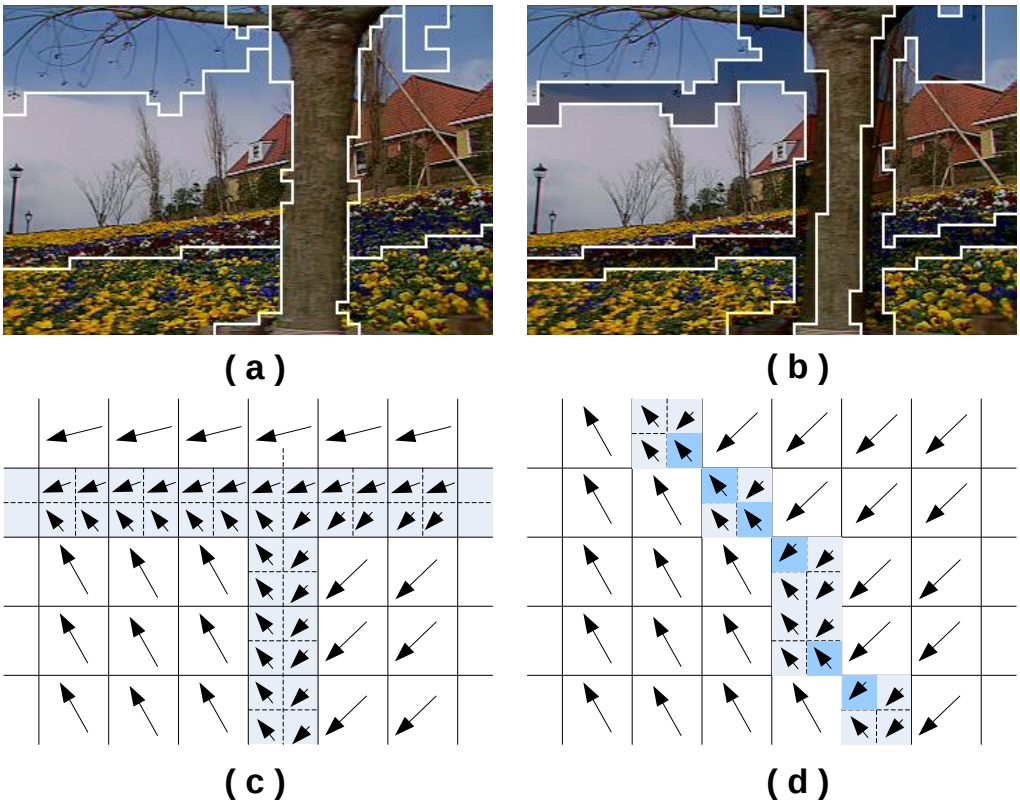


Figure 5.1: (a) the block-based boundary after motion segmentation; (b) identified boundary blocks for variable-block-size motion prediction; (c) MV prediction for 4×4 sub-blocks when each 4×4 sub-block has 8×8 neighbours from only one region; (d) MV prediction for 4×4 sub-blocks when some 4×4 sub-blocks (shown in darker blue) have 8×8 neighbours from different regions.

Boundary blocks are split into smaller blocks (in our case, 8×8 blocks are split into four 4×4 blocks), and a MV is assigned to each of the smaller blocks as illustrated in Fig. 5.1 (c) and (d), based on the following criteria.

If all 8×8 (internal) blocks neighbouring a given 4×4 sub-block come from the same region, then the centroid MV of that region is assigned to the 4×4 sub-block. This is illustrated in Fig. 5.1 (c).

If a 4×4 sub-block has 8×8 neighbours that come from different regions, then the MV distance is first calculated between these regions' centroid MVs and the MV of the

parent 8×8 block (before splitting). The closest centroid MV is assigned to the 4×4 sub-block. This is illustrated in the right part of Fig. 5.1 (d), where sub-blocks with 8×8 neighbours from a single region are shown in light gray, while sub-blocks with 8×8 neighbours from different regions are shown in blue. Specifically, if the 4×4 sub-block is surrounded by regions labelled $R \in \{1,2,\dots,N\}$, each having its centroid vector $\mathbf{MV}_{cent}^{R}$, and $\mathbf{MV}_{parent}$ is the MV of its parent 8×8 block, then its MV, denoted $\mathbf{MV}_{4\times4}$, is found as

$$\mathbf{MV}_{4\times4} = \arg\min_{1\le R\le N} \left\| \mathbf{MV}_{cent}^{R} - \mathbf{MV}_{parent} \right\| \qquad \text{Equation 5.2}$$

5.1.2. Preliminary Frame Synthesis

Once the segmentation is complete and MVs are assigned to each block, we move all blocks of the last received frame onto the future frame along the predicted MVs. In this way, we synthesize a preliminary version of the future frame. At this point, some areas of the synthesized frame may have multiple blocks landing on them – we call these areas *overlapped areas*. Other areas may remain empty, if no block lands on them – we call these *empty areas*. We need to decide which pixel values will be written into overlapped and empty areas. These decisions are made in the two post-processing steps described below.

5.1.3. Post Processing

In order to balance computational complexity and visual quality, we combine spatial interpolation and boundary matching to process *empty areas* and *overlapped areas.* We further distinguish two types of (empty or overlapped) areas: "thin" areas are those whose width or height does not exceed N pixels, while "thick" areas are those whose both width and height exceed N pixels ($N = 3$ in our experiments). Simple spatial interpolation is applied to thin areas to save the computation and memory access, while boundary matching is applied to thick areas to achieve better visual quality.

5.1.3.1. Overlapped Area

For the "thin" overlapped area, we apply simple averaging of all candidate pixel values. Let there be N blocks overlapping a certain area and let OV_k denote the k-th

block. The pixel value at location (x, y) in the overlapped area is assigned to be the average of corresponding pixel values in each of the overlapping blocks:

$$P(x, y) = \frac{1}{N} \cdot \sum_{k=1}^{N} OV_k(x, y) \qquad \text{Equation 5.3}$$

Once all thin overlapped areas are processed, we are left with thick overlapped areas whose height and width exceed 3 pixels. One such area is shown in Fig. 5.2.

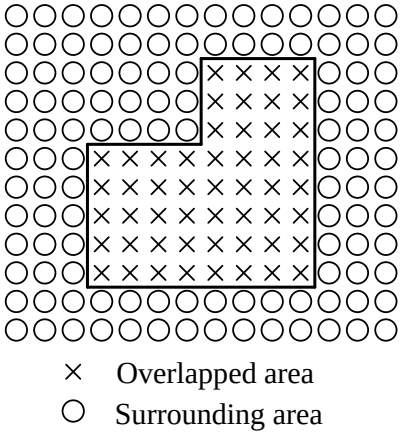


Figure 5.2: Thick overlapped area.

These areas will be filled by pixel values from the block that fits the best into the surrounding area. To decide which block fits the best, we employ boundary matching by computing the mean square difference between the boundary pixels of candidate blocks, and the boundary pixels of the surrounding area.

Let $OV_k(x, y)$ be the pixel at location (x, y) in the k-th overlapping block. Let B be the set of boundary pixels of the overlapped area, and for each $(x, y) \in B$, let $n(x, y)$ be the value of the neighbouring pixel across the boundary, in the surrounding area. If a pixel has multiple neighbours across the boundary (e.g., corner pixel), we first check if there are any top/bottom neighbours, then left/right neighbours, then top-left/top-right, etc. The first neighbour found in this search is taken as $n(x, y)$. The best matching block

OV_{best} is the one whose square difference from the surrounding area along the boundary is the smallest, as in Eq. 5.4. Pixels from this block are used to fill the thick overlapped area.

$$OV_{best} = \text{argmin}_k \sum_{(x,y)\in B} \|OV_k(x,y) - n(x,y)\|^2 \qquad \text{Equation 5.4}$$

5.1.3.2. Empty Area

Thin empty areas are filled using simple linear spatial interpolation [97] in the direction in which the empty area is no more than 3 pixels wide. An illustration of a thin empty area whose height is 3 pixels is shown in Fig. 5.3. Let $P(x,y)$ be the pixel value we wish to determine in an empty area, and let $P_1(x_1,y_1)$ and $P_2(x_2,y_2)$ be two of its nearest neighbours in the appropriate direction in the surrounding area. In the situation depicted in Fig. 5.3, P_1 and P_2 are above and below P, so in this case $x_1 = x_2 = x$. The pixel in the empty area is linearly interpolated as:

$$P(x,y) = \left(1 - \frac{h_1}{H}\right) P_1(x_1,y_1) + \left(1 - \frac{h_2}{H}\right) P_2(x_2,y_2) \qquad \text{Equation 5.5}$$

where h_1 and h_2 are the distances from P to P_1 and P_2, respectively, and $h_1 + h_2 = H$.

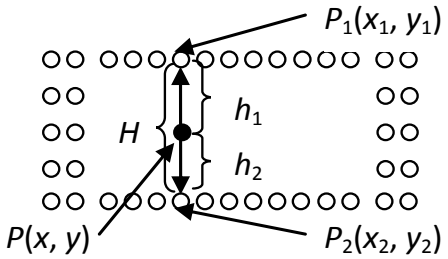


Figure 5.3: Filling thin empty areas.

Simple linear interpolation works reasonably well for thin empty areas, but tends to produce excessive blurring when applied to thick empty areas. Therefore, we adopt a

more sophisticated method for filling thick empty areas based on boundary matching [98], [99].

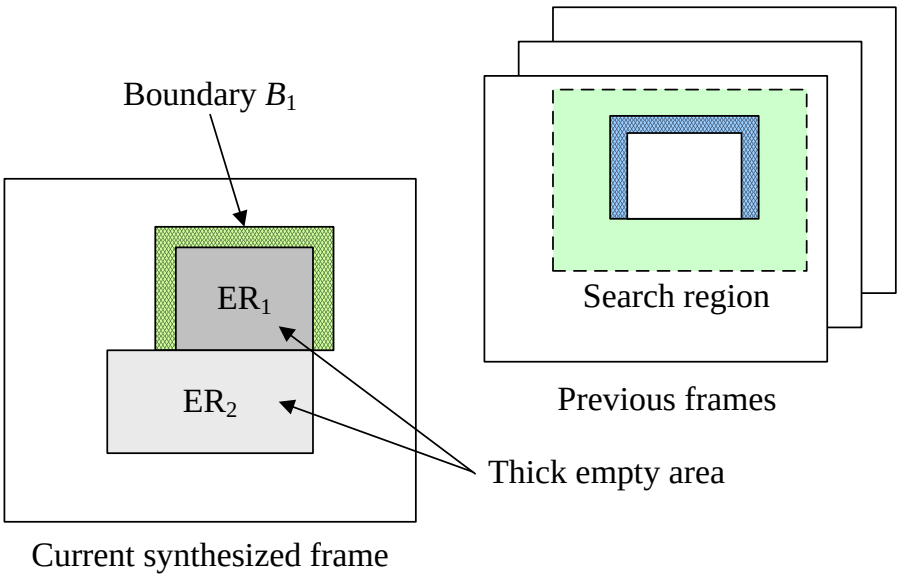


Figure 5.4: Filling thick empty areas.

An example of a thick empty area is shown in Fig. 5.4. First, we divide each thick empty area into rectangular regions, which we call *empty rectangles* (ERs), and label them ER_1, ER_2, …, ER_N. We fill ERs in sequence, starting with ER_1 and ending with ER_N. For each ER we extract the boundary pixels from the surrounding area and use them for boundary matching in previous frames. Let B_n be the set of boundary pixel coordinates for ER_n. Denote the current frame as P, and previous K frames as $P_1, P_2, \ldots, P_K$. We will search in each of the previous K frames over an area of size $X \times Y$ pixels for the best matching boundary (in our experiments, $X = Y = 32$). This boundary is found in frame P_k, offset by (dx, dy) from its position in the current frame, where

$$(k, dx, dy) = \arg \min_{k=1,2,\ldots,K} \left\{ \min_{\substack{|dx| \le X/2 \\ |dy| \le Y/2}} D^{P_k} \right\} \qquad \text{Equation 5.6}$$

$$D^{P_k} = \sum_{(x,y)\in B_n} \|P(x,y) - P_k(x+dx, y+dy)\|^2 \qquad \text{Equation 5.7}$$

Once Eq. 5.6 is solved and the best matching boundary is found, we copy the corresponding rectangle from P_k to fill ER_n. At this point, ER_n is removed from the list of empty rectangles, and we continue with ER_{n+1}. The pixels of ER_n may now become boundary pixels for the remaining empty rectangles.

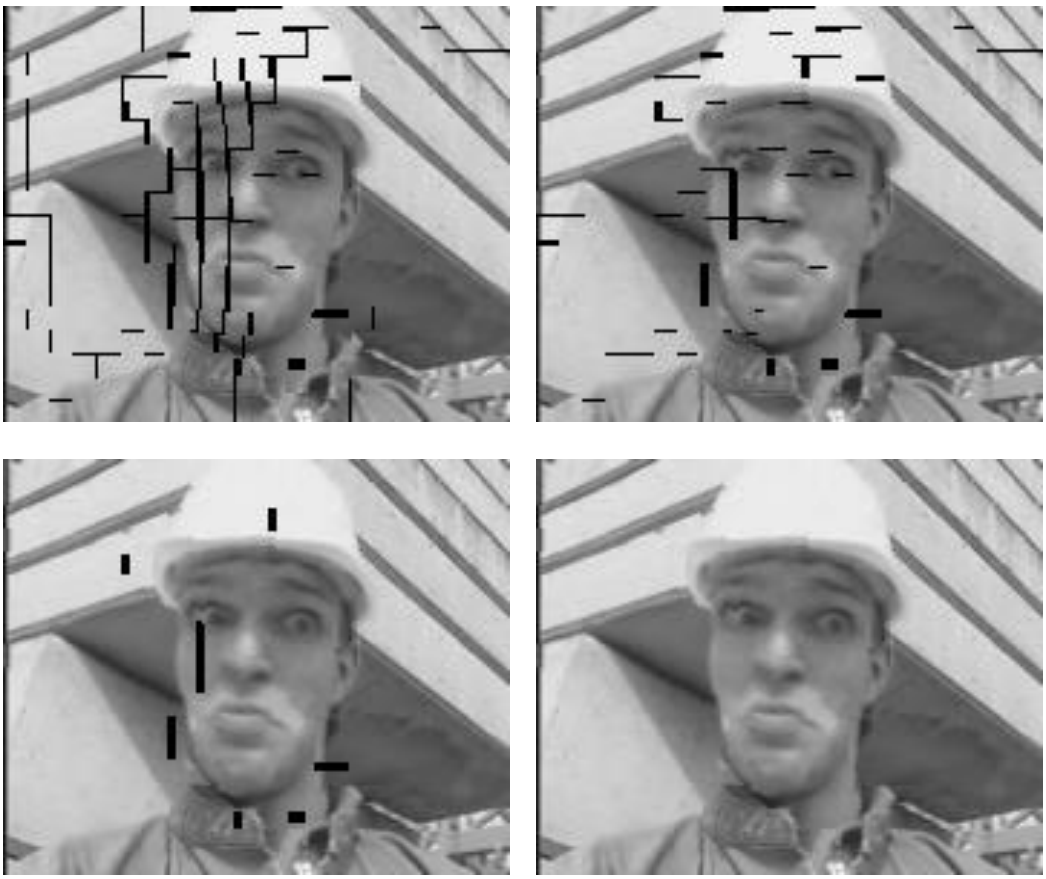


Figure 5.5: Illustration of empty area post-processing: intermediate frame after post-processing of overlapped areas (top left); after filling thin vertical empty areas (top right); after filling thin horizontal empty areas (bottom left); final frame after filling thick empty areas (bottom right).

The filling is performed in raster scan order. An example of empty area filling is shown in Fig. 5.5. The figure shows a frame passing through the steps of empty area post-processing. The top left image shows an intermediate frame after overlapped area processing. Thin vertical empty areas are filled first (top right), followed by thin horizontal empty areas (bottom left). The final predicted frame, obtained after filling thick empty areas, is shown in the bottom right of Fig. 5.5.

5.2. Object-Based Synthesis using ordinal depth

The frame synthesis method in the previous section does not explicitly consider occlusions. In this section, we present another block-based method based on the observation that when two regions move towards the same location, the region closer to the view point will occlude the region further away from the view point. This method is present in our previous work [92].The background region(s) are considered to have the largest distance to the view point. Fig. 5.6 shows how to multiplex foreground objects with background objects together and obtain a new frame.

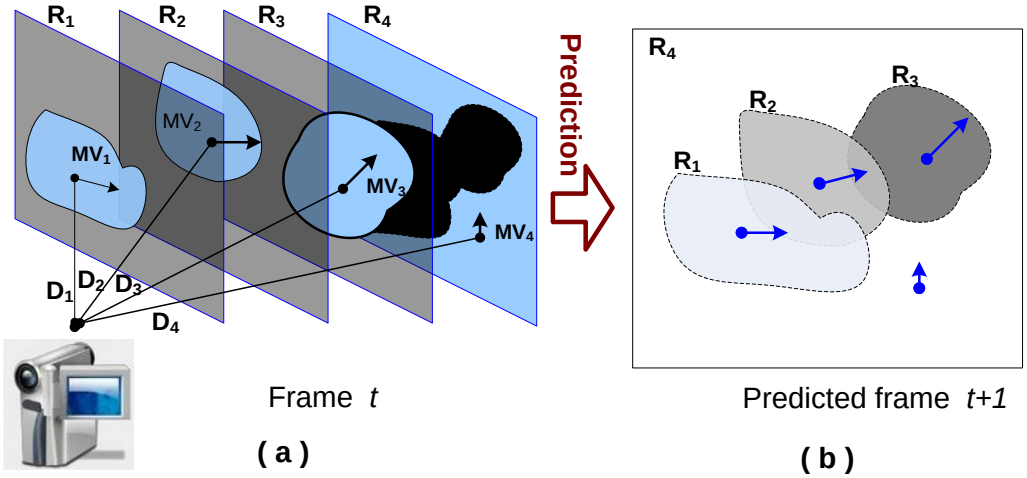


Figure 5.6: Content-based frame composition

In the example in Fig. 5.6, there are three foreground regions (R_1, R_2, R_3), and one background region (R_4) in frame t. Each foreground region is associated with a MV,

and located at a layer which has a depth value D_i. Global motion is estimated for background region (R_4). Frame prediction using ordinal depth is illustrated in Fig. 5.6. Let D_i and $\mathbf{MV}_i$ be, respectively, the distance of region R_i from the camera plane and its predicted motion vector. Then the future frame $t+1$ is synthesized by moving R_i along $\mathbf{MV}_i$, and placing the pixels of R_i in front of R_j if $D_i < D_j$. Assuming R_1 is the closest region to the camera, and R_4 is the furthest ($D_4 > D_3 > D_2 > D_1$), predicted frame at $t+1$ will be as shown in Fig. 6(b).

If two regions (R_i and R_j) move towards the same location, they may overlap in a certain area of the frame. This area can be determined as

$$r_{i,j} = R_i^{t \to t+m} \cap R_j^{t \to t+m} \qquad \text{Equation 5.8}$$

where $R_n^{t \to t+m}$ represents the translation of region R_n from the image plane at time t onto the image plane at time $t+m$ along its motion vector. The preliminary frame $\mathbf{f}_{t+m}$ is synthesized as

$$\mathbf{f}_{t+m} = T_{GP}\left(R_1^{t \to t+m} \oplus R_2^{t \to t+m} \oplus R_3^{t \to t+m} \oplus R_{Background}\right) \qquad \text{Equation 5.9}$$

where T_{GP} represents the predicted perspective global motion model, and the operation $\oplus$ represents the aggregation of pixels from different regions, defined as follows

$$R_i \oplus R_j(x,y) = \begin{cases} R_i(x,y), & (x,y) \in R_i \backslash R_j \\ R_j(x,y), & (x,y) \in R_j \backslash R_i \\ R_k(x,y), & (x,y) \in R_i \cap R_j \end{cases} \qquad \text{Equation 5.10}$$

where $k = \arg\min_{k \in \{i,j\}}(D_i, D_j)$. Region depths D_i and D_j determine which region gets occluded during frame synthesis. In the example in Fig. 5.6, region R_1 will be “in front of” the other three regions (R_2, R_3, R_4), while the background region R_4 will be “behind” other regions (R_1, R_2, R_3).

In Chapters 2–4, we have proposed methods to obtain preliminary object features — such as classification of foreground and background, contour, global motion, and texture — from compressed video. In this section, we outline the major functional blocks on how to establish the spatial relationship between objects, predict moving trajectories and composite a frame under various circumstances.

5.2.1. Ordinal Depth Estimation

Depth estimation is conceptually used to find the spatial information of the objects or regions in a video sequence [83-85]. In this work, region depth order describes the relative distance of the moving regions from the camera and is used to arbitrate the overlapped areas in the synthesized frame when multiple regions move onto the same location, as in Eq. 5.9 and 5.10. The process is much harder when the depth information needs to be estimated from a 2D video than a multi-view video sequence, since most of the depth information is lost when a 3D real life scene is projected onto a 2D image plane by a video acquisition system. Here we use a simple and reasonably effective method whose basic premise is that the more similar the region's motion is to the camera motion, the further the region is from the camera.

First we classify moving regions into two types: *background regions*, whose motion is similar to camera motion, and *foreground regions*, whose motion is distinct from camera motion. A relatively simple approach for classification is to measure the discontinuity of the motion field across a region's boundaries. The idea was first studied in [86], where the authors observed that the discontinuity of the motion field when crossing from a foreground region to a background region is usually stronger than when crossing from one background region to another.

The classification starts with the calculation of the boundary discontinuity value BD_R for each region. The initial input into the classification consists of coarsely segmented regions. Let $b_1, b_2, \ldots, b_N$ be the N blocks along the outer boundary of a region. The boundary discontinuity value BD_R is calculated as:

$$BD_R = \frac{1}{N} \cdot \sum_{b=b_1}^{b_N} \left(\text{argmax}_{k=1,\cdots,4} \left(\Delta_k(b) \right) \right) \qquad \text{Equation 5.11}$$

where $\Delta_k(b) = \left\| \mathbf{MV}_{NE_m} - \mathbf{MV}_{NE_n} \right\|$ is one of four directional gradient components shown in Fig. 5.7, and $\mathbf{MV}_{NE_m}$ denotes the MV of the m-th neighbouring block.

Similarly to [86], the classification is carried out as follows:

Step 1. Calculate BD_R for each region, as explained above;

Step 2. Sort all regions according to their BD_R;

Step 3. Classify the region with the largest BD_R as a foreground region, and accumulate overall foreground region size;

Step 4. Repeat Step 3) until the accumulated foreground region size reaches TH_{fg} of the total frame size, where $TH_{fg} = 30\%$ in this work;

Step 5. Flag the remaining regions as background regions.

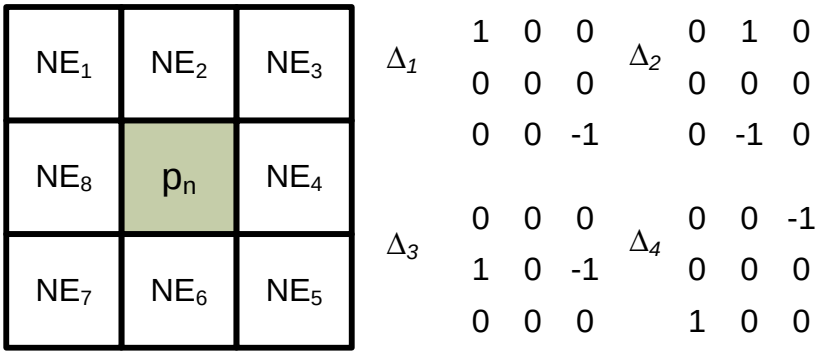


Figure 5.7: [Left]: 3×3 neighbourhood of a boundary block, [Right]: four directional gradient components.

Camera motion vector $\mathbf{PMV}_{CAM}$ is predicted as the average of predicted MVs of all background regions. After this, we estimate the relative distance D_i of region i from the camera as the inverse of the magnitude of the difference of its predicted motion from the predicted camera motion

$$D_i = \|\mathbf{PMV}_i - \mathbf{PMV}_{CAM}\|^{-1} \qquad \text{Equation 5.12}$$

If the motion vector difference is zero, the particular region is considered to be very far from the camera — "at infinity." This simple estimate of the relative distance of a region from the camera will not be correct in some cases, but it has proven to be reasonably good in our experiments. After estimating all the D_i values, a future frame is formed by moving the regions along the predicted motion vectors, as in Eq. 5.9, with an additional rule that foreground regions always locate "in front of" background regions regardless of their D_i.

5.2.2. Region Motion Prediction

Predicting how one region moves from the current frame to the next frame is a crucial step for final frame composition. Since regions are segmented in the first place as the collections of blocks that move in a coherent fashion, we predict motion on a region basis and assign the same motion vector to all pixels in a particular region. In this way, we can suppress, to some extent, the noise present in encoded MVs and keep the region intact over the predicted moving trajectory.

State-of-the-art motion prediction methods include linear prediction [87-88], least-squares prediction [89], statistical prediction [90] and Kalman filtering [91]. In this thesis, we predict the region motion from its causal past using the first order least-squares prediction method [89]. Let $\mathbf{MV}_{cent}^{R_n}(t)$ denote the centroid MV of region R_n in frame t. Through region tracking, we can collect MV centroids of region R_n for the past K frames. The first order least-squares predictor for the motion of region R_n up to the frame $t+m$ is:

$$\mathbf{PMV}_{n_n}(m) = \mathbf{a}_0 + \mathbf{a}_1 m \qquad \text{Equation 5.13}$$

where the closed-form solution for $\mathbf{a}_0$ and $\mathbf{a}_1$ is

$$[\mathbf{a}_0, \mathbf{a}_1] = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{Y} \qquad \text{Equation 5.14}$$

where $\mathbf{Y}$ is a $K \times 2$ matrix containing the last K MV centroids of R_n as rows, and $\mathbf{H}$ is a $K \times 2$ matrix given by

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 2 & \cdots & K \end{bmatrix}^T \qquad \text{Equation 5.15}$$

We used $K = 8$ in our experiments.

The accuracy of least squares motion prediction is analyzed in our earlier work [92]. Figure 5.8 illustrates the prediction error introduced for region R_n. Here, $\mathbf{MV}_{t+i}$ denotes the "true" centroid MV of region R_n obtained from encoded MVs, and $\mathbf{PMV}_{t+i}$ denotes the predicted MV obtained from Eq. 5.13.

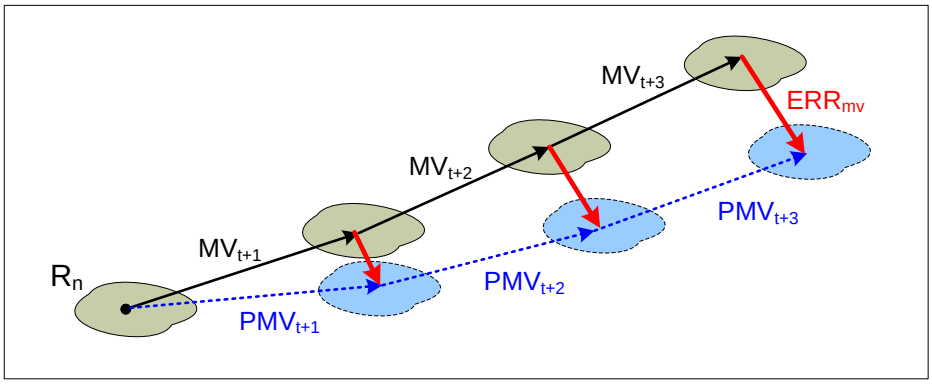


Figure 5.8: Region motion prediction error

Figure 5.9 shows segmented foreground object “tree trunk” in *Flower Garden*, the magnitude of the true centroid of MVs for the tree, along with the magnitudes of **PMV**s for 1, 2, and 3 frames ahead, for the first 50 frames of the sequence. Tracking allows us to collect the “true” MVs of this region. We also predict the motion of this region for 1, 2 and 3 frames ahead. The comparison of the magnitude of the true centroid of MVs and **PMV**s is shown in the middle part of Fig. 5.9, while the prediction error magnitude is shown in the bottom part of Fig. 5.9. Note that, as the frame index increases, the tree moves to the left part of the frame and further away from the camera, so the intensity of its motion decreases from about 6 pixels per frame down to about 4 pixels per frame. Predicted MVs follow this behaviour. The prediction error magnitude is less than 1 pixel when predicting the motion for one frame ahead, increases up to about 1.5 pixels for two frames ahead and then up to about 2 pixels for three frames ahead.

5.2.3. Frame Synthesis and Post Processing

A preliminary version of the future frame is synthesized by moving all regions onto the future frame along the predicted MVs in the order of their depth values Eq. 5.9 and 5.10. At this point, some areas of the synthesized frame may remain empty if no region lands on them. As in Section 5.1.3.2, for empty areas whose width or height does not exceed 3 pixels, we apply simple linear spatial interpolation along the direction where the thickness of the area is smallest. Empty areas whose both width and height exceed 3 pixels are filled via boundary matching. Two frames are used as reference frames for boundary matching: one previously reconstructed frame and one “background frame,” which is generated by gradually replacing the foreground regions with the newly appearing background regions based on region classification [93].

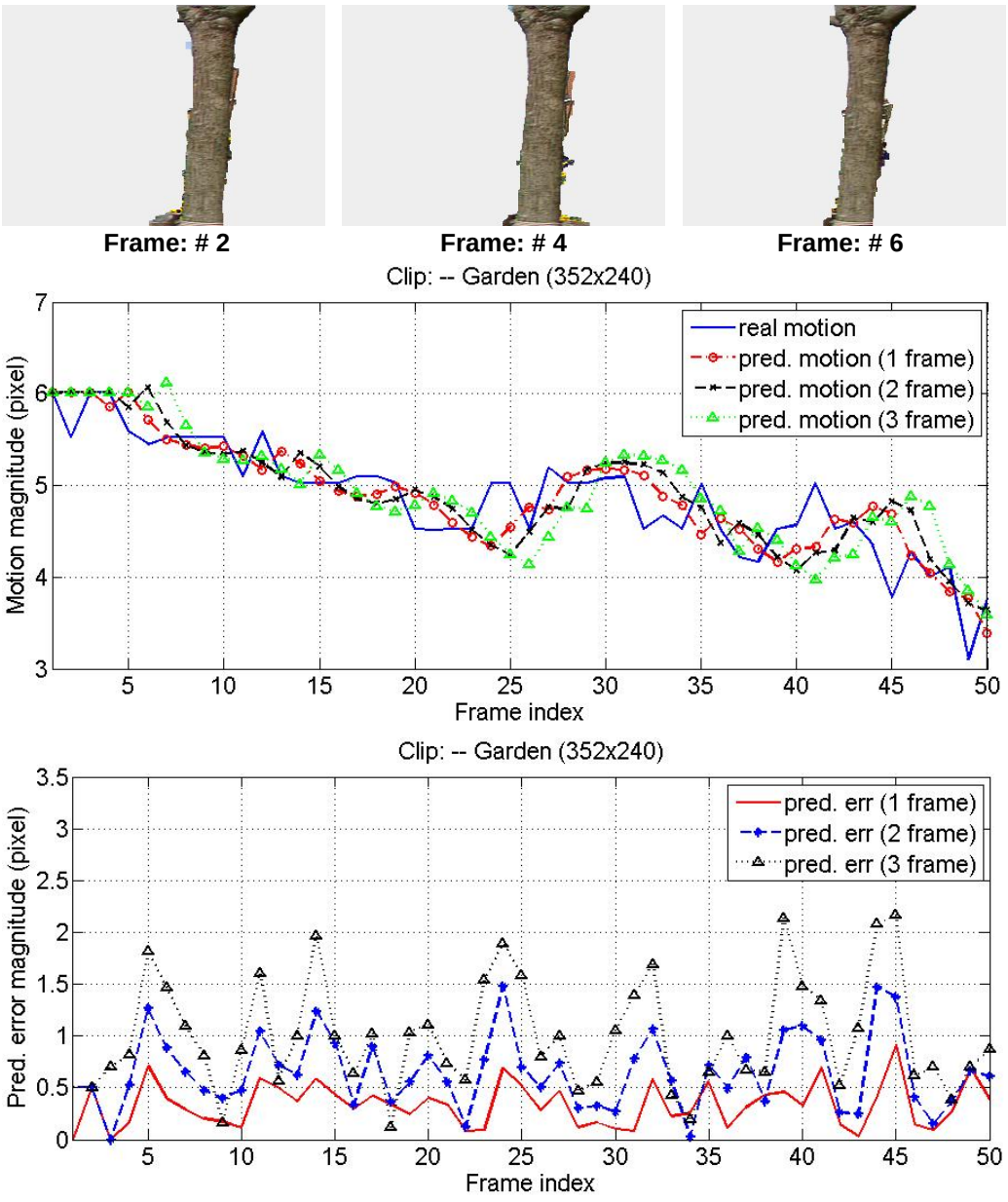


Figure 5.9: Motion of the tree in *Flower Garden.* [Top]: segmented object, [Middle]: magnitude of the true and predicted motion, [Bottom]: prediction error magnitude

5.3. Pixel-Based Synthesis Using LMMSE Criterion

This approach sets apart from the previous two object-based frame synthesis approaches by fusing temporally and spatially synthesized frames on a pixel basis. Spatial synthesis plays an important role especially when the resolution of the target frame is different from the original frame resolution. The fusion weights are derived by assessing the accuracy of each synthesis method on a block-by-block basis and then combining them in a Linear Minimum Mean Squared Error (LMMSE) fashion. The inputs to the synthesis are from three sources: 1) a frame temporally synthesized using local motion (LM), i.e. MVs, 2) a frame temporally synthesized using global motion (GM), and 3) a spatially synthesized frame. The details of frame interpolation in various application scenarios are given in Chapter 6. The main idea is to determine a set of weights $\{w_S, w_{TL}, w_{TG}\}$ for each pixel in the target frame. The weights reflect the accuracy of each synthesis method in a particular block; the method with the highest accuracy will end up with the highest weight in that block. The final target frame is obtained as a linear combination of the three preliminary target frames on a block-by-block basis. This approach has been presented in our previous work [100] [101].

The weights on temporal and spatial synthesis should reflect the level of quality impact on the target frame. In this section, we present the Linear Minimum Mean Square Error (LMMSE) combination of temporal and spatial interpolation. Let $\hat{\mathbf{f}}_{TL}$, $\hat{\mathbf{f}}_{TG}$, and $\hat{\mathbf{f}}_S$ be column vectors containing the pixels from a particular block of the preliminary target frames generated using LM, GM and spatial synthesis, respectively, and let $\hat{\mathbf{f}}_{target}$ be the column vector containing the pixels of that same block in the final target frame. Then we have:

$$\hat{\mathbf{f}}_{target} = w_{TL}\hat{\mathbf{f}}_{TL} + w_{TG}\hat{\mathbf{f}}_{TG} + w_S\hat{\mathbf{f}}_S \qquad \text{Equation 5.16}$$

where the weights satisfy $0 \le w_{TL}, w_{TG}, w_S \le 1$ and $w_{TL} + w_{TG} + w_S = 1$. Let $\mathbf{f}$ be the column vector containing the pixels of the same block from the "correct" (ideal) target frame and let $\mathbf{e}_{TL}, \mathbf{e}_{TG}, \mathbf{e}_S$ denote the column vectors containing prediction errors generated using LM, GM and spatial synthesis, respectively, so that:

$$\hat{\mathbf{f}}_{TL} = \mathbf{f} + \mathbf{e}_{TL} \quad \text{Equation 5.17}$$

$$\hat{\mathbf{f}}_{TG} = \mathbf{f} + \mathbf{e}_{TG} \quad \text{Equation 5.18}$$

$$\hat{\mathbf{f}}_{S} = \mathbf{f} + \mathbf{e}_{S} \quad \text{Equation 5.19}$$

We want to determine the weights $\{w_{TL}, w_{TG}, w_S\}$ that will minimize the Mean Squared Error (MSE) between the correct target frame $\mathbf{f}$ and the synthesized target frame $\hat{\mathbf{f}}_{target}$:

$$E\left[\left(\mathbf{f} - \hat{\mathbf{f}}_{target}\right)^T \left(\mathbf{f} - \hat{\mathbf{f}}_{target}\right)\right] = E[\left(\mathbf{f} - w_{TL}\hat{\mathbf{f}}_{TL} + w_{TG}\hat{\mathbf{f}}_{TG} + w_S\hat{\mathbf{f}}_{\mathrm{S}}\right)^T \cdot \left(\mathbf{f} - w_{TL}\hat{\mathbf{f}}_{TL} + w_{TG}\hat{\mathbf{f}}_{TG} + w_S\hat{\mathbf{f}}_{\mathrm{S}}\right)] \quad \text{Equation 5.20}$$

Assuming that interpolation errors $\mathbf{e}_{TL}, \mathbf{e}_{TG}, \mathbf{e}_S$ are zero-mean and uncorrelated, Eq. 5.20 can be simplified to:

$$E\left[\left(\mathbf{f} - \hat{\mathbf{f}}_{target}\right)^T \left(\mathbf{f} - \hat{\mathbf{f}}_{target}\right)\right] = w_{TL}^2 E[\mathbf{e}_{TL}^T \mathbf{e}_{TL}] + w_{TG}^2 E[\mathbf{e}_{TG}^T \mathbf{e}_{TG}] + w_S^2 E[\mathbf{e}_S^T \mathbf{e}_S] \quad \text{Equation 5.21}$$

To minimize equation 5.21 under the constraint $w_{TL} + w_{TG} + w_S = 1$, we form the Lagrangian:

$$J = w_{TL}^2 E[\mathbf{e}_{TL}^T \mathbf{e}_{TL}] + w_{TG}^2 E[\mathbf{e}_{TG}^T \mathbf{e}_{TG}] + w_S^2 E[\mathbf{e}_S^T \mathbf{e}_S] - \lambda(w_{TL} + w_{TG} + w_S - 1) \quad \text{Equation 5.22}$$

and then set the partial derivatives with respect to w_{TL}, w_{TG}, w_S, to zero. This leads to:

$$w_{TL}E[\mathbf{e}_{TL}^T\mathbf{e}_{TL}] = w_{TG}E[\mathbf{e}_{TG}^T\mathbf{e}_{TG}] = w_S E[\mathbf{e}_S^T\mathbf{e}_S] \qquad \text{Equation 5.23}$$

Combining equation 5.23 with the condition that $w_{TL} + w_{TG} + w_S = 1$, we finally find the LMMSE weights as:

$$w_{TL} = \frac{E[\mathbf{e}_{TG}^T\mathbf{e}_{TG}]E[\mathbf{e}_S^T\mathbf{e}_S]}{G}$$

$$w_{TG} = \frac{E[\mathbf{e}_S^T\mathbf{e}_S]E[\mathbf{e}_{TL}^T\mathbf{e}_{TL}]}{G} \qquad \text{Equation 5.24}$$

$$w_S = \frac{E[\mathbf{e}_{TL}^T\mathbf{e}_{TL}]E[\mathbf{e}_{TG}^T\mathbf{e}_{TG}]}{G}$$

where G is given by:

$$\begin{aligned} G = {} & E[\mathbf{e}_{TG}^T\mathbf{e}_{TG}]E[\mathbf{e}_S^T\mathbf{e}_S] \\ & + E[\mathbf{e}_S^T\mathbf{e}_S]E[\mathbf{e}_{TL}^T\mathbf{e}_{TL}] \\ & + E[\mathbf{e}_{TL}^T\mathbf{e}_{TL}]E[\mathbf{e}_{TG}^T\mathbf{e}_{TG}] \end{aligned} \qquad \text{Equation 5.25}$$

Note that the LMMSE weights depend on the interpolation errors $\{\mathbf{e}_{TL}, \mathbf{e}_{TG}, \mathbf{e}_S\}$ between the correct and synthesized blocks in the target frame. Since the correct blocks are not known (otherwise, if the frame compositor knew what the correct frame looks like, there would be no need to synthesize it), these error vectors need to be estimated for each block. We explain how this can be in done in two applications, super-resolution and predictive decoding, in Chapter 6.

6. Applications

Making use of motion modelling and object segmentation in a conventional video decoding system, we are able to retrieve object information from a compressed video stream and simultaneously observe its movement. The ultimate goal in this work is to composite a frame with a desired temporal and spatial scale based on collected object information. From an application point of view, the combined motion modelling and object segmentation can be broadly applied in three types of scenarios with a standard video decoding system.

Approach 1). ***Predicting*** a future frame using causal processing. For example, a future frame can be composited ahead of time in video communications, so that network jitter can be mitigated, or a missing frame can be concealed.

Approach 2). ***Interpolating*** a frame using noncausal processing. For example, the frame rate of the original sequence can be up-converted for a variety of video applications.

Approach 3). ***Resizing*** a frame up or down using available video data. For example, a super-resolution (SR) frame can be generated for rendering video to a large screen.

6.1. Frame Prediction

The basic idea behind frame prediction in video communications is illustrated in Fig. 6.1. Using the received video data, future frames are predicted and displayed before they arrive at the decoder. We refer to this process as “predictive decoding,” since the input is compressed video and the output is a video frame that hasn’t yet arrived at the decoder. This technique can be useful in many video applications, such as delay reduction in video communications [52], playout control in video streaming [107] and the concealment of whole-frame losses [88] [102] [108].

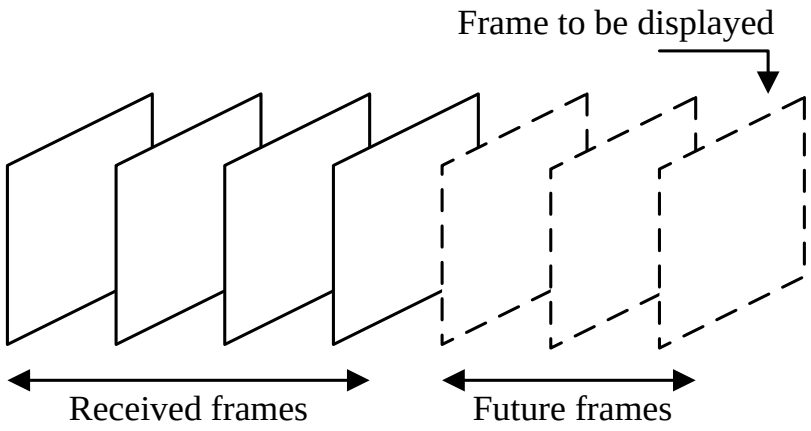


Figure 6.1: Perceived delay reduction by predictive decoding.

Conventional frame prediction approaches, such as motion extrapolation [88] [102], heavily rely on the motion vectors (MVs) extracted from the compressed video stream. However, MVs encoded into the bitstream are usually estimated at the encoder with the aim of improving compression efficiency by minimizing the energy of the prediction residual and do not always represent true motion [5]. These modeling and estimation inaccuracies become more pronounced when prediction is performed and cause annoying artefacts in predicted frames, such as background shaking and spatial discontinuities. As will be discussed in this section, we propose three predictive decoding approaches to address these issues. These approaches take different combinations of motion prediction, object segmentation and frame composition, as presented in Chapters 2–5.

1) Predictive decoding using moving region segmentation
2) Predictive decoding using ordinal depth of moving regions
3) Predictive decoding using GME and motion reliability

6.1.1. Background on Predictive Decoding

Though video delay reduction through predictive decoding does not seem to have been explored much in the literature thus far, except for our work [52], [92], [93],

[100], several algorithms have been proposed for whole-frame concealment [87], [102 – 106], and these algorithms can also be used to perform frame prediction at the decoder for the purpose of reducing the end-to-end delay. This observation motivates us to compare our work against the previously proposed methods for whole-frame concealment, which can be broadly divided into two categories: pixel-based and block-based. Below we review a representative method from each category.

6.1.1.1. Pixel-based Whole Frame Concealment

Based on the optical flow theory, a whole-frame concealment algorithm is proposed for video streaming applications in [87] [102 – 106]. Assume the frame at time t is lost. Operating in the pixel domain, a constant velocity motion model is adopted to project the last correctly received frame onto the missing frame by the following steps.

1) Constructing a forward motion vector (FMV) for each pixel in the last received frame (i.e. frame $t-1$) from the backward MVs and coding modes of frames from $t-2$ to $t-L-1$ using the constant velocity model;

2) Spatially regularizing and smoothing the FMV field of frame $t-1$ using the vector median filter;

3) Reconstructing the missing frame by projecting pixels from frame $t-1$ into the missing frame t in half-pixel resolution, so that each pixel of frame $t-1$ contributes a 2×2 pattern, and then averaging pixels if more than one of them has landed on the same position (overlapped areas);

4) Scanning the remaining missing pixels in frame t and estimating these missing pixels using the median of the neighbouring available pixels (empty areas);

5) Filtering and downsampling the reconstructed frame by a factor of two in both the horizontal and vertical directions in order to convert it from half-pixel to full-pixel representation.

One drawback of a pixel-based, whole-frame concealment is its computational complexity. According to [103], predicting one CIF frame using this algorithm might take 3–6 seconds on a conventional desktop machine. Applying a computationally-intensive

algorithm to reduce the end-to-end delay is likely to be counter-productive, especially in cases where more than one frame needs to be predicted, because it would place additional burden on the decoder during the real-time operation. Furthermore, averaging pixels in overlapped areas and median filtering of pixels in empty areas may be able to mitigate to some extent the problems of empty and overlapped areas, but may also lead to objectionable artefacts in the parts of the frame with complex motion or texture.

6.1.1.2. Block-based Whole Frame Concealment

Block-based frame concealment was proposed by Baccichet *et al*. [103 – 104] based on the same optical flow concepts, but requiring lower complexity. This algorithm consists of the following steps:

1) Determining the reference frame with a significant number of inter coded blocks (usually frame $t-1$), and constructing a FMV for each pixel in frame $t-1$;

2) Predicting pixel-wise FMV from frame $t-1$ into pixels in the missing frame t;

3) Computing the statistics (mean and variance) of FMVs for each 16×16 and 4×4 block in frame t;

4) If multiple MVs from frame $t-1$ land on the same 16×16 block, then the variance and number of these MVs are compared to the thresholds. If the variance of these MVs is sufficiently low, and their number is sufficiently high, the mean MV is assigned to this block. Similar processing is applied to 4×4 blocks;

5) Reconstructing the final frame using motion compensation and loop filtering as in H.264/AVC.

Compared to a pixel-based approach, a block-based approach has a significantly lower computational complexity because MV processing is block-based rather than pixel-based. However, by averaging neighbouring MVs, MV recovery in a block-based approach may disrupt the spatio-temporal relationships among the MVs of neighbouring blocks and thus may lead to blocking artefacts. Moreover, the algorithm requires extra effort on the encoder side to produce better MVs and one additional step to estimate MVs for intra-coded blocks at the decoder [105].

6.1.2. *Predictive Decoding Using Moving Region Segmentation*

In [93], we proposed a predictive decoding algorithm using moving region segmentation, as illustrated in Fig. 6.2. It consists of three steps:

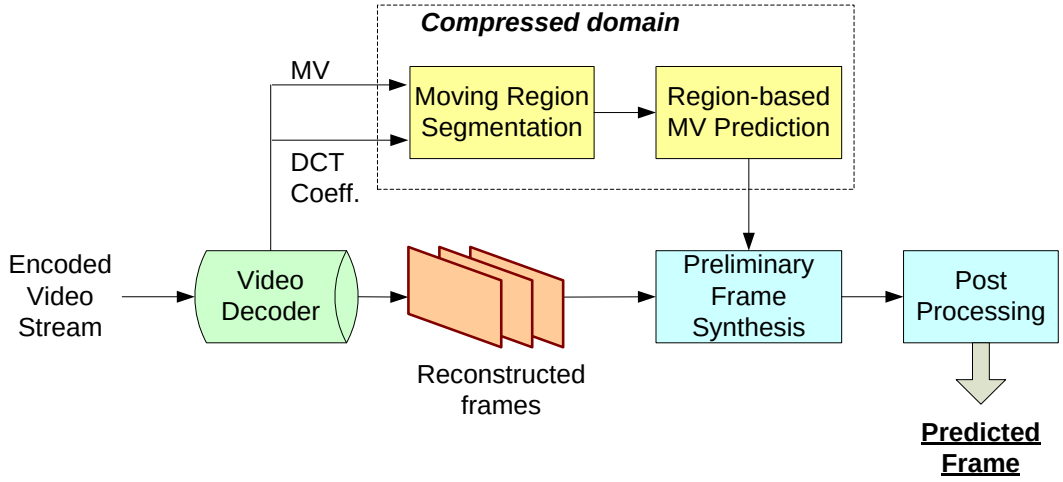


Figure 6.2: Functional block diagram of region-based predictive decoding.

Step 1. MVs and prediction residuals from the compressed bitstream are used to segment moving regions in frame $t-1$. The detail of motion segmentation was elaborated on in Chapter 3, Section 3.1.

Step 2. A region-based MV prediction estimates the motion trajectory of the moving regions in frame $t-1$ towards frame t, as presented in Chapter 5, Section 5.1.1. Predicting motion trajectories is a crucial component of frame prediction. By employing region-based MV prediction in the predictive decoding system, we are able to reduce the effects of MV noise and unnatural motion artefacts in the predicted frame, such as image background shaking and spatial discontinuities. Motion trajectory prediction based on moving regions offers a way to enforce motion field homogeneity within regions and may, to some extent, reduce MV estimation noise.

Step 3. A preliminary version of the future frame is synthesized by moving blocks from frame $t-1$ to frame t along the predicted MVs. The post processing unit mitigates the spatial discontinuities (empty and overlapped areas) of the preliminary future frame. In particular, linear interpolation and motion re-estimation via boundary matching are employed to process empty areas, while pixel averaging and boundary-based block selection are combined to process overlapped areas, as elaborated on in Chapter 5, Section 5.1.

We have implemented this approach in the XviD MPEG-4 decoder and tested its performance on a number of sequences with varying levels of motion complexity. Six sequences — *Carphone*, *Flower Garden*, *Foreman*, *Singer*, *Mother & Daughter*, and *Miss America* — are used for performance demonstration, each at three different frame rates: 30, 15 and, 7.5 frames per second (fps). Each sequence was encoded using the IPPP GOP structure. Unless otherwise stated, QCIF sequences were encoded at 128 kbps and CIF/SIF sequences at 512 kbps. On the decoder side we tested prediction of up to 3 frames ahead using the three prediction methods below.

Three methods are included in algorithm evaluation:

- **Method-1** ("frame copy") is the simplest prediction method, where the last received frame is taken directly as the predicted frame for playout. This method is used as the basis (anchor) against which other methods are judged, as in [88].
- **Method-2** is our implementation of the block-based whole frame concealment approach [88] in the XviD MPEG-4 decoder.
- **Method-3** is the proposed region-based predictive decoding.

6.1.2.1. Performance Evaluation on Prediction Depth

Figure 6.3 shows how video quality measured by PSNR in dB depends on how far ahead we predict. Each PSNR plot shows the PSNR averaged over the entire sequence when predicting 1, 2 or 3 frames ahead. All these results correspond to 30 fps sequences. From the PSNR plots we can observe the following. First, as expected, PSNR decreases as prediction goes further. The quality degradation depends on the motion activity level and texture pattern in the sequence. *Flower Garden* has a high

degree of motion and the most complicated texture pattern among our test sequences. When predicting one frame ahead the PSNR drops by about 3dB for **Method-2** and **Method-3**, and more than 10dB for **Method-1**. On sequences with relatively high motion complexity, like *Flower Garden*, *Singer*, *Carphone* and *Foreman*, our **Method-3** outperforms **Method-2** by 0.5 – 2.5dB, while both these methods outperform **Method-1** by about 1 – 7dB, depending on the sequence. For sequences with relatively low motion, like *Miss America* and *Mother & Daughter*, sophisticated frame prediction employed in **Method-2** and **Method-3** does not provide as much improvement as for high-motion sequences and simple methods like **Method-1** do reasonably well, so the difference between the three methods is rather small.

Figure 6.4 gives a visual quality comparison between **Method-2** and **Method-3** when predicting up to three frames ahead using the *Flower Garden* sequence. Three frames (#11 – #13) are predicted by **Method-2** and **Method-3** using the frames up to frame #10. We can observe that the predicted frame quality deteriorates as the prediction depth increases for both methods. However, frames produced by **Method-3** are visibly better than those produced by **Method-2**. This is due to a significantly reduced level of blocking artefacts, especially in the moving region (tree trunk in this case) where all blocks have consistent motion, which is exploited in our **Method-3.**

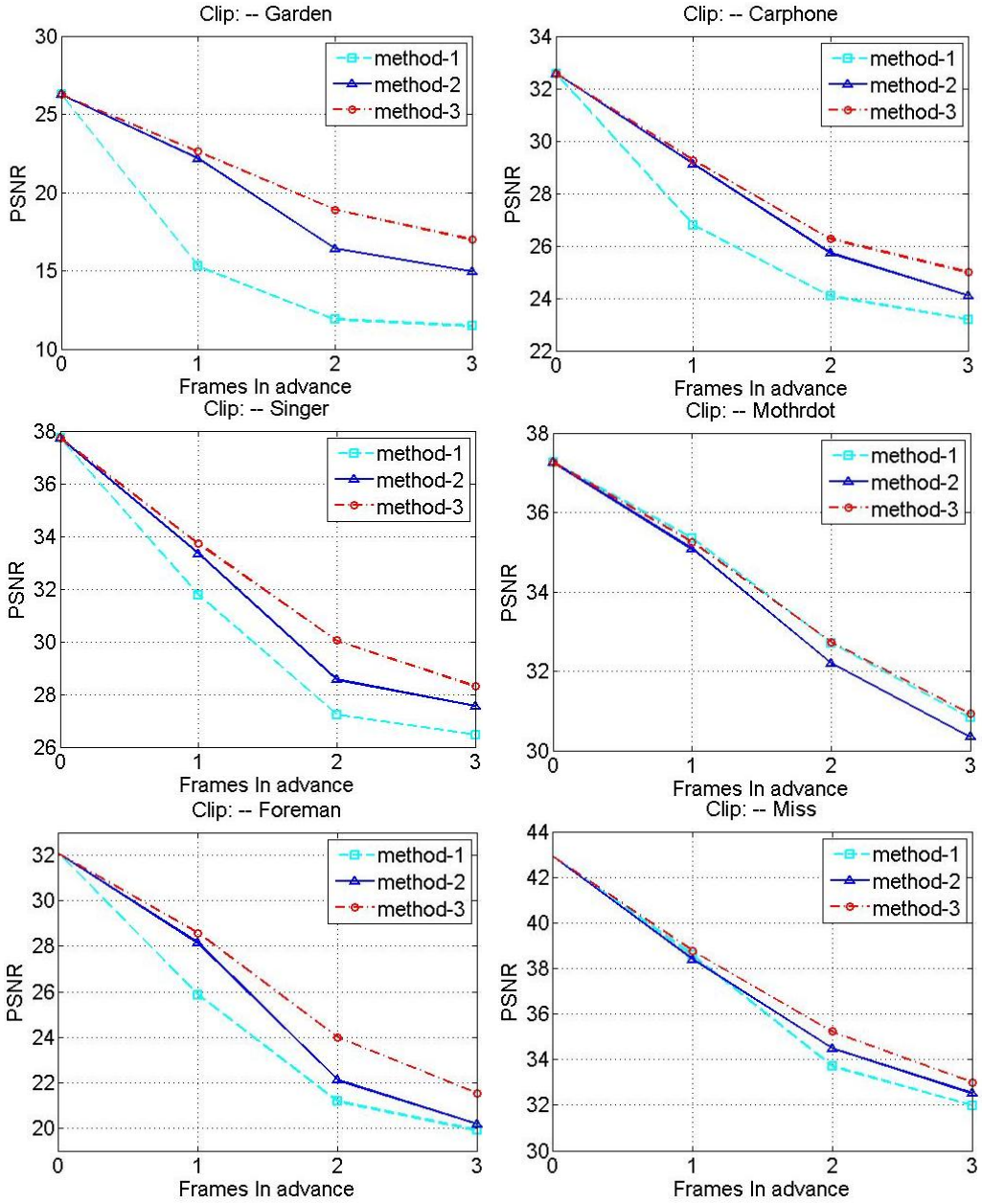


Figure 6.3: Prediction performance in PSNR (dB).

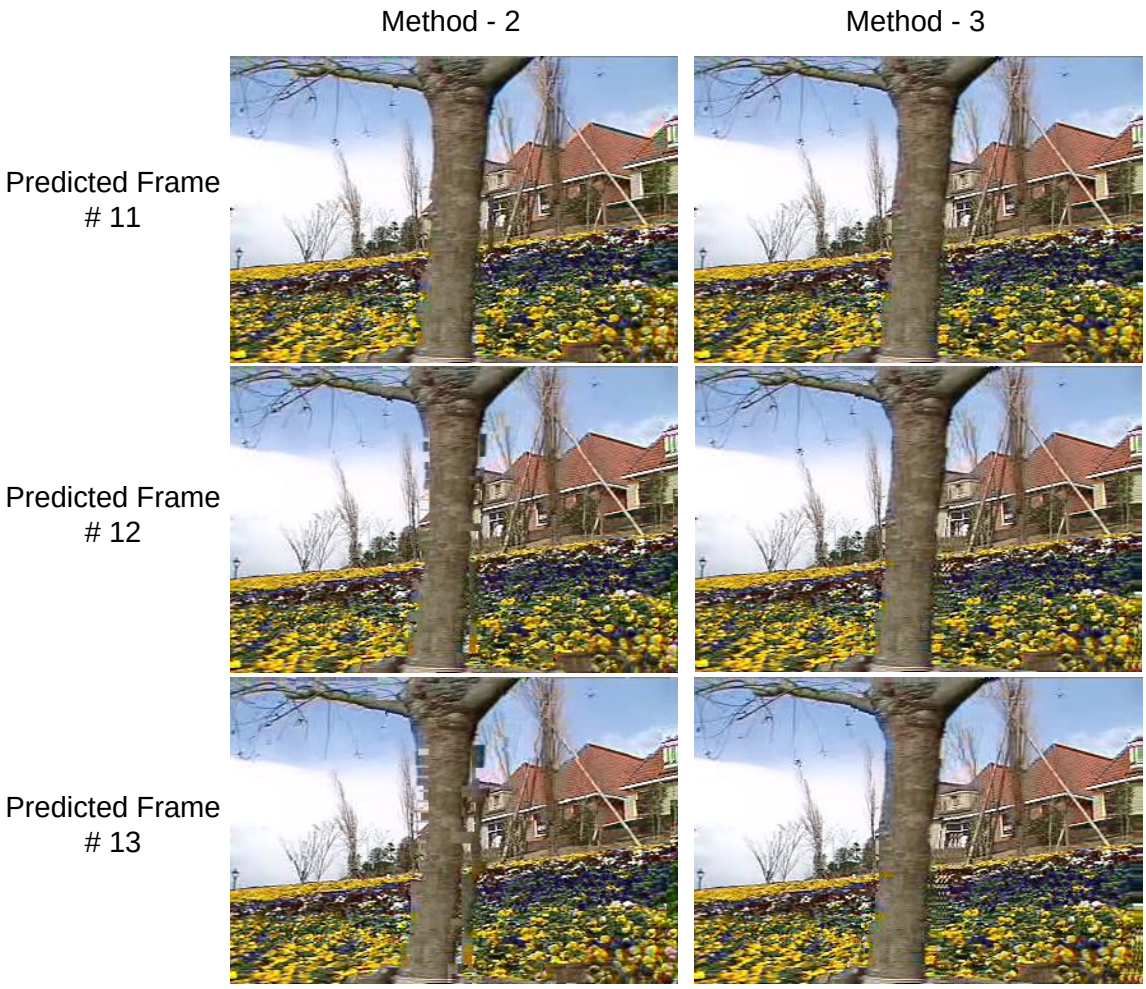


Figure 6.4: Visual comparison for frame prediction assessment. [Left]: predicted frames (Method-2), PSNR (22.33dB, 16.95dB, 15.63dB), [Right]: predicted frames (Method-3), PSNR (22.47dB, 18.70dB, 16.78dB).

6.1.2.2. Performance Evaluation on Frame Rate and Bitrate

Figures 6.5 and 6.6 show the prediction performance of **Method-3** at different frame rates and bitrates for the two sequences *Foreman* (QCIF) and *Flower Garden* (SIF). Other sequences follow similar trends. Frame rates in these experiments are set to 30fps, 15fps and 7.5fps, and bitrates are set to 128kbps, 256kbps and 512kbps for

QCIF resolution, with 512kbps, 1.024mbps, and 2.048mbps for SIF resolution. Prediction depth goes up to 3 frames ahead in both test scenarios.

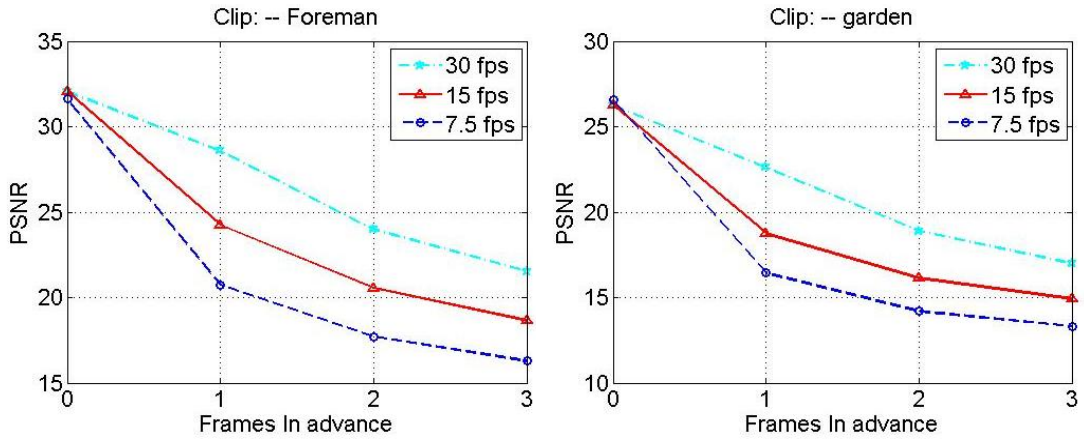


Figure 6.5: Prediction performance at different frame rates.

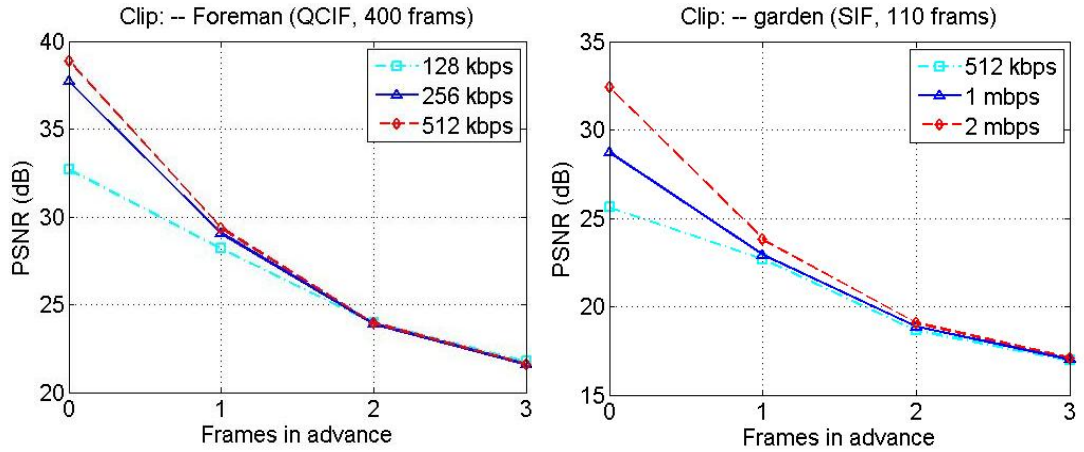


Figure 6.6: Prediction performance at different bitrates.

From Fig. 6.5 we can observe that the prediction is better at higher frame rates, which is consistent with our expectations, since the temporal spacing between the frames is smaller. It is interesting to observe that the PSNR of predicted frames is approximately determined by the total amount of delay reduction or, in other words, by

the prediction depth measured in seconds rather than frames. For example, predicting one frame ahead with 15fps *Foreman* gives us similar PSNR as when predicting two frames ahead with 30fps on *Foreman*. From Fig. 6.6 we observe that a higher bitrate brings a 1–1.5dB higher PSNR when predicting one frame ahead. However, as prediction goes further the PSNR difference between different bitrates reduces. This is caused by the fact that prediction error increases with prediction depth and dominates the over quantization error controlled by the bitrate.

6.1.3. Predictive Decoding Based on Ordinal Depth of Moving Regions

The problem with the prediction method presented in the Section 6.2 is that it does not consider occlusion. Near object boundaries, especially when predicting several frames ahead, one can notice background sometimes occluding the foreground regions. An example is the bottom right frame in Figure 6.4, where the blue color of the sky starts overtaking the tree trunk near the left boundary of the tree. The method discussed in this section performs predictive decoding based on ordinal depth of moving regions, thereby trying to alleviate occlusion problems. This method has been presented in [92].

First, motion information (i.e. MVs) is extracted from the compressed video stream and region segmentation is performed to obtain coherently moving regions in the scene. Moving region segmentation is carried out using our previously developed coarse-to-fine segmentation algorithm from [27], which is also discussed in Section 3.3. We first use MVs from the compressed bitstream to coarsely segment moving regions by a combination of iterative *k*-means clustering and a motion consistency model. This step operates fully in the compressed domain. After that, the current frame is decoded and the boundaries of moving regions are refined using the colour and edge information from the decoded frame, as discussed in Section 3.4.4.

Second, the ordinal depth of each region (i.e. how far it is from the camera) is estimated using the procedure described in Section 5.2.1. Finally, a least-squares motion predictor is applied to each region (Section 5.2.2) and the future frame is synthesized by moving the regions along the predicted trajectories sequentially according to their ordinal depth. This way, the moving regions are kept intact and they

are placed onto the future frame in the correct order. For the details of these two steps please refer to Chapter 5, Section 5.2.

6.1.3.1. Objective and subjective quality evaluation

We integrated the proposed predictive decoding method into the popular Xvid MPEG-4 video decoder (www.xvid.org), and tested it on several test sequences with varying motion content, resolution, and bitrate. The experimental conditions are summarized in Table 6.1. We tested prediction of up to 3 frames ahead using the following three prediction methods:

- **Method-1:** "frame copy" as the anchor method,
- **Method-2:** the block-based predictive decoding based on whole frame concealment approach proposed in [88].
- **Method-3** is our proposed method.

Both **Method-1** and **Method-2** are the same ones used in previous section, and we adopt them here for a consistent performance evaluation.

Table 6.1: Experimental setup for predictive decoding evaluation

Codec	**Xvid MPEG-4**
Development env.	C/C++ in Microsoft Visual Studio 2005
Prediction depth	Up to 3 frames ahead of time
Encoded frame rate	30 frames per second (fps)
Coding pattern	IPPP…
Video resolution	CIF (352×288), SIF (352×240), QCIF (176×144)
Test sequences	*Mother & Daug*hter, *Flower Garden*, *Foreman*, *Table Tennis*, *Tempete*, *Hall Monitor*, *Coastguard, Football*
Encoded bitrate	512 kbps (SIF and CIF), 256 kbps (QCIF)
CPU	Intel Pentium D - 3.0 GHz
RAM	2 GB

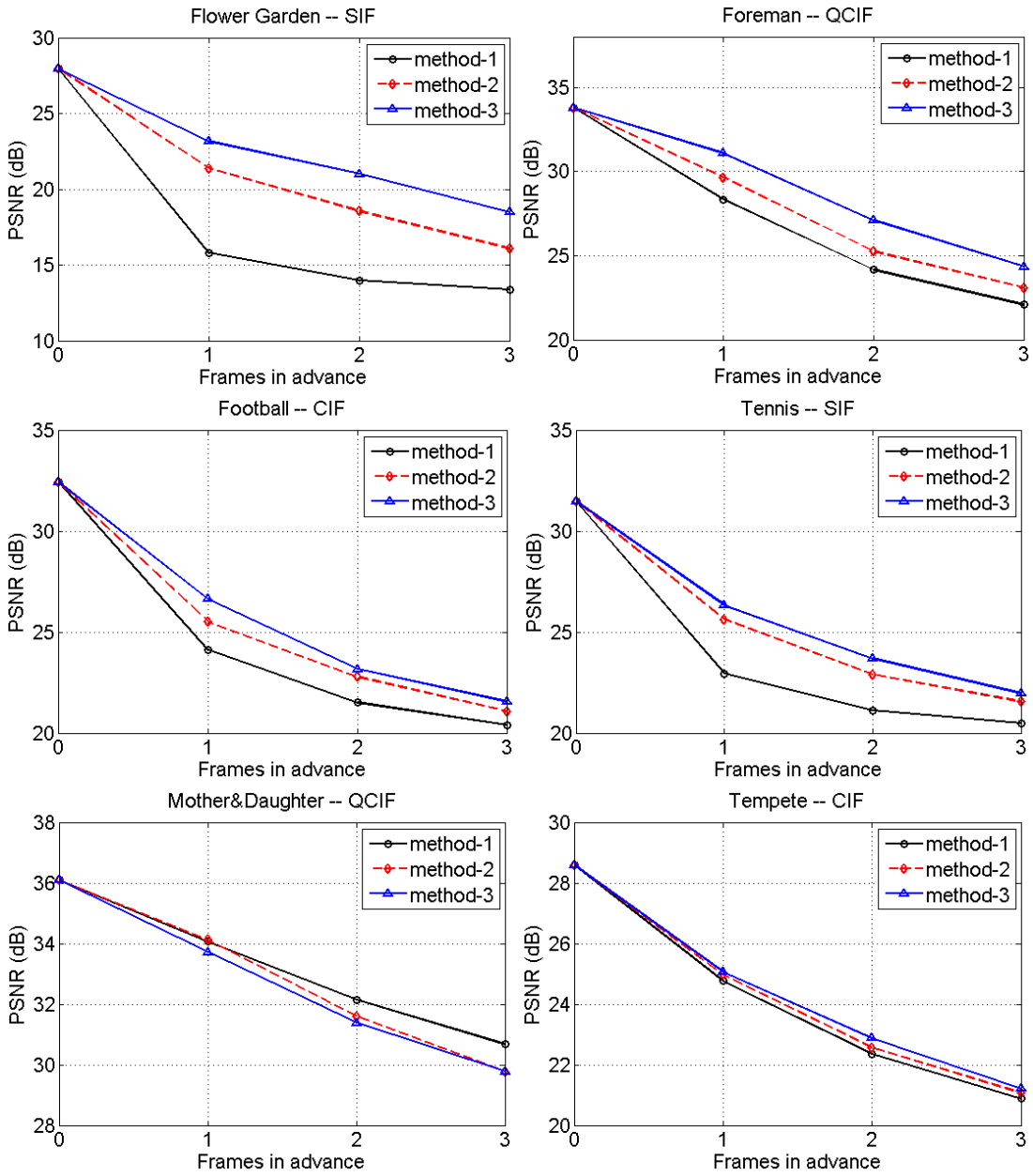


Figure 6.7: The PSNR vs. prediction depth.

Figure 6.8: [Top to Bottom]: Predicted frames #7 to #9 for sequence *Flower Garden* (using video data up to frame #6). [Left]: Method-2, PSNR (22.93dB, 19.41dB, 17.79dB), [Right]: Method-3, PSNR (23.98dB, 21.11dB, 19.27dB).

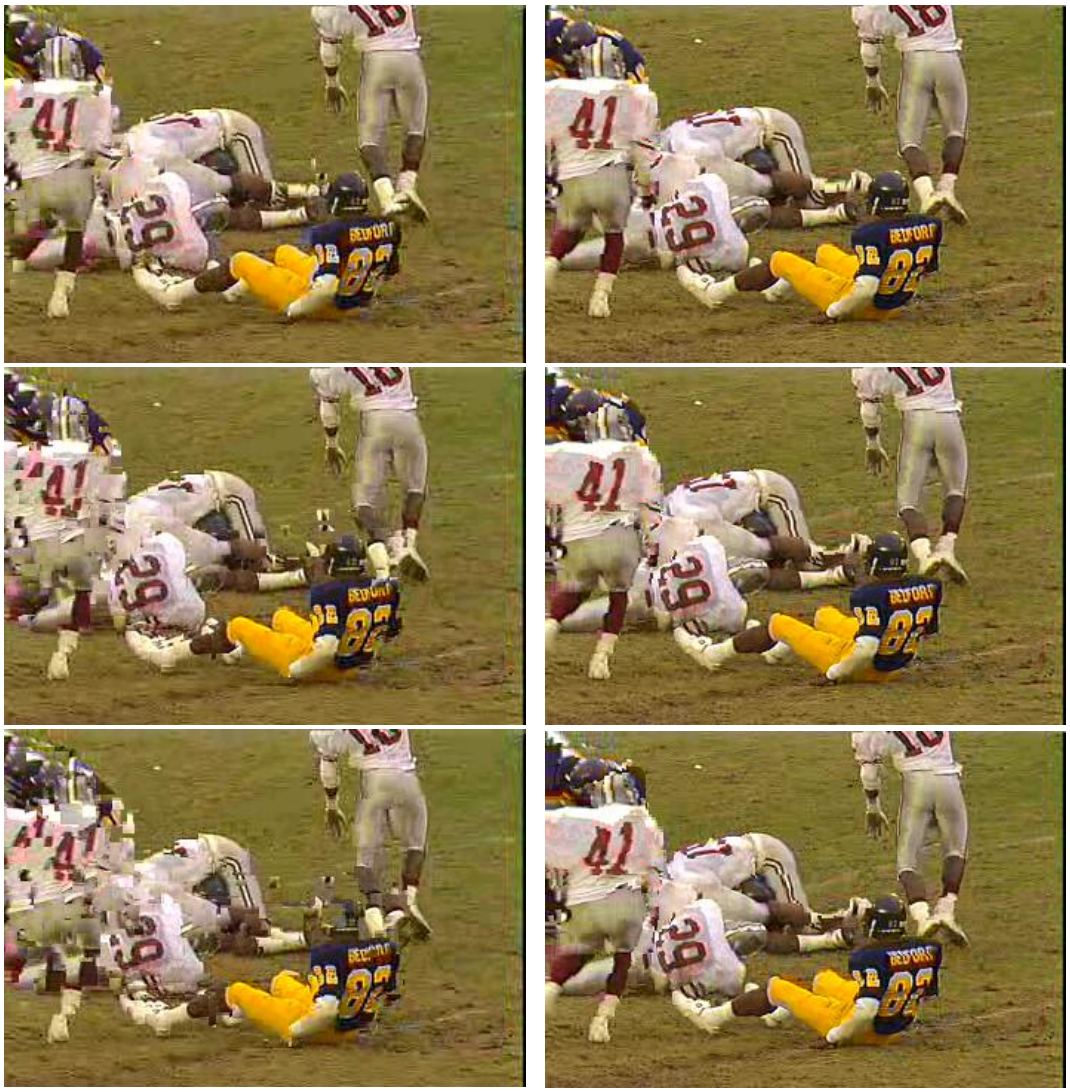


Figure 6.9: [Top to Bottom]: Predicted frames #41 to #43 for sequence *Football* (using video data up to frame # 40). [Left]: Method-2, PSNR (25.87dB, 23.38dB, 20.71dB), [Right]: Method-3, PSNR (26.84dB, 23.99dB, 21.70dB).

We now investigate the quality of predicted frames in terms of PSNR in dB. Figure 6.7 shows the PSNR comparison of the three prediction methods for six test sequences: *Table Tennis, Flower Garden*, *Foreman*, *Football*, *Tempete* and *Mother & Daughter*. We can observe from these comparisons that the proposed **Method-3** outperforms other methods for sequences with medium or high motion, as with *Foreman*

(up to 4dB over **Method-1** and 1.5 dB over **Method-2**) and *Flower Garden* (up to 7dB over **Method-1** and 2.1dB over **Method-2**). For sequences with low motion, such as *Mother & Daughter*, even a simple frame copy (**Method-1**) seems to work well and the other two methods do not offer any prediction advantage. On the other hand, the sequence *Tempete* contains zooming, which is not well-matched to the block translation motion model and causes many inaccurate MVs in the compressed bitstream. This hurts **Method-2** and **Method-3**, which rely on encoded MVs for prediction, so their gain over **Method-1** is relatively small in this case as well.

The subjective quality of predicted frames is illustrated in Fig. 6.8 (*Flower Garden*) and Fig. 6.9 (*Football*). We compare prediction results for up to 3 frames ahead using **Method-2** and **Method-3**. As expected, the further ahead we predict, the lower the quality of the predicted frames for both methods, especially near the boundaries of the moving regions. Nevertheless, the subjective quality advantage of **Method-3** is evident. **Method-2** suffers from significant blocking artefacts near moving region boundaries and from occlusions of the foreground object by background blocks. This is less of a problem for **Method-3**, due to its region-based predictive approach and ordinal depth ordering.

6.1.4. Predictive Decoding Using GME and Motion Reliability

The two frame prediction methods presented in the previous sections did not consider camera motion. Hence, in this section, we describe a frame prediction framework that incorporates global (camera) motion modeling and estimation. This way, we can perform frame prediction using both local motion (LM) and global motion (GM) extrapolation. The task of a frame prediction system is to determine:

1) Global and local motion for each pixel;
2) Weights on global and local motion per pixel for target frame synthesis.

Thus, the composition of a one-step target frame $\hat{\mathbf{f}}$ can be formulated as

$$\hat{\mathbf{f}}(x,y,t+1) = w_{LM}T\big(\mathbf{f}(x+LMV^X, y+LMV^Y, t)\big) + w_{GM}T\big(\mathbf{f}(x+GMV^X, y+GMV^Y, t)\big) \quad \text{Equation 6.1}$$

where $\mathbf{f}(x, y, t)$ denotes a pixel value at (x, y) in frame t. LMV^X and LMV^Y are, respectively, the *X*- and *Y*- components of local MV at (x, y), while GMV^X and GMV^Y are *X*- and *Y*- components of MVs synthesized from a global motion model. T is the temporal interpolator to generate the preliminary target frame. Weights w_{LM} and w_{GM} determine the influence of local and global temporal interpolation on the resulting predicted frame.

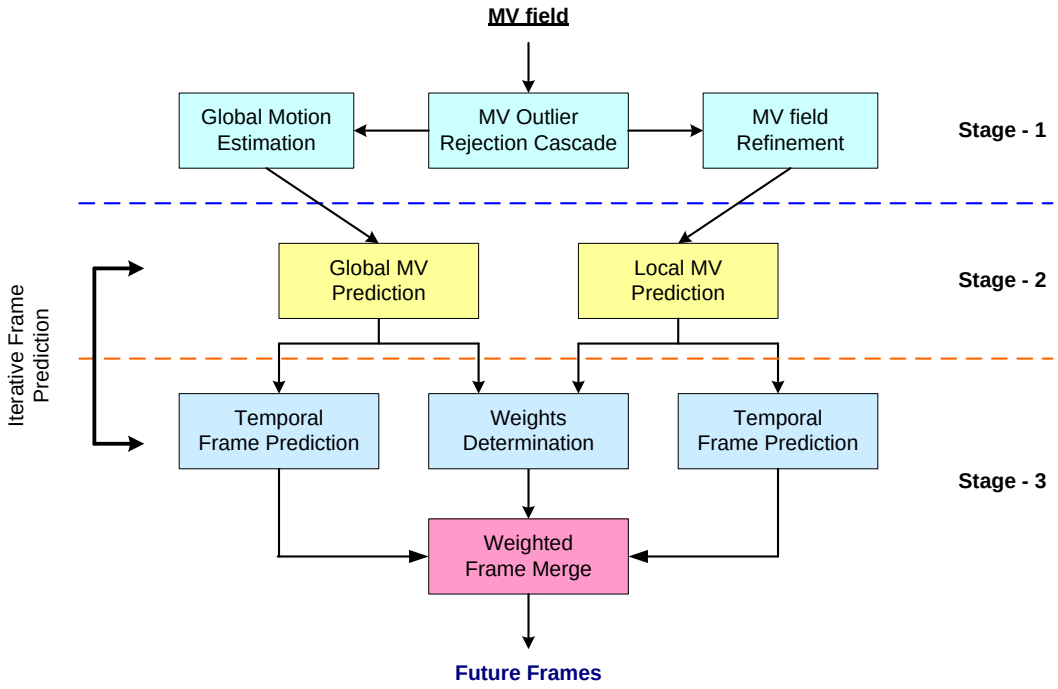


Figure 6.10: Predictive decoding system using GME and motion reliability

Based on this concept, we present a predictive decoding framework that decouples local and global motion and consists of three processing stages. Global motion (GM) estimation and motion reliability analysis are the key components in the first stage, where we estimate global motion parameters and refine the MV field. In the second stage, we predict local and global motion for the target frame and determine corresponding weights based on the Linear Minimum Mean Square Error (LMMSE) criterion. Finally, in the third stage, a temporal interpolator T is applied to compose two

target frames that are linearly combined to form the final predicted target frame. The proposed MCPD framework is schematically shown in Fig. 6.10.

Stage-1, **-2**, and -**3** represent the procedures employed in the proposed frame prediction. These stages can be further iterated to compose multiple target frames, as shown in Fig. 6.10. By re-using the MV field produced from **Stage-1** we can iteratively perform **Stage-2** and **Stage-3** to predict more frames. We describe each stage in detail in the following sections.

6.1.4.1. MV Reliability and Global Motion Estimation

MVs extracted from the video bitstream are usually estimated by a block-matching algorithm in the encoder. For many reasons the MV estimates may fail to represent the true motion, e.g. poor texture of the block that an MV is associated with or boundary blocks of a moving object. An MV field can be smoothed out through common filtering techniques, such as a 2D median filter, however, a drawback of such a filter is that good MVs might be contaminated from surrounding noisy MVs. In this section, we apply MV reliability analysis before MV refinement so that we can distinguish reliable MVs from unreliable ones and selectively feed them into different processing modules.

MV reliability is rated through a cascade of three MV filters that were proposed in [28], and described in Section 2.2.4. Each filter in the cascade constructs a particular MV pattern in a 3×3 neighbourhood and examines the reliability in terms of relative MV magnitude difference and phase difference. We then conduct a binary classification through a hard thresholding, and a certain percentage of the least reliable MVs are identified as MV outliers. In this section, we target on 20% of the least reliable MVs as the outliers and the remaining 80% of MVs as the inliers. Once MVs are classified, inliers are fed into a GM estimator, while outliers are further subjected to an MV refinement. As an example, in Fig. 6.11 we show the MV field extracted from the bitstream along with identified MV outliers with the overall inlier percentage thresholds set to 70%, 80% and 90%, where the outliers are marked in a darker colour while the inliers are marked in a brighter colour. Frame #2 of *Flower Garden* is used in this example.

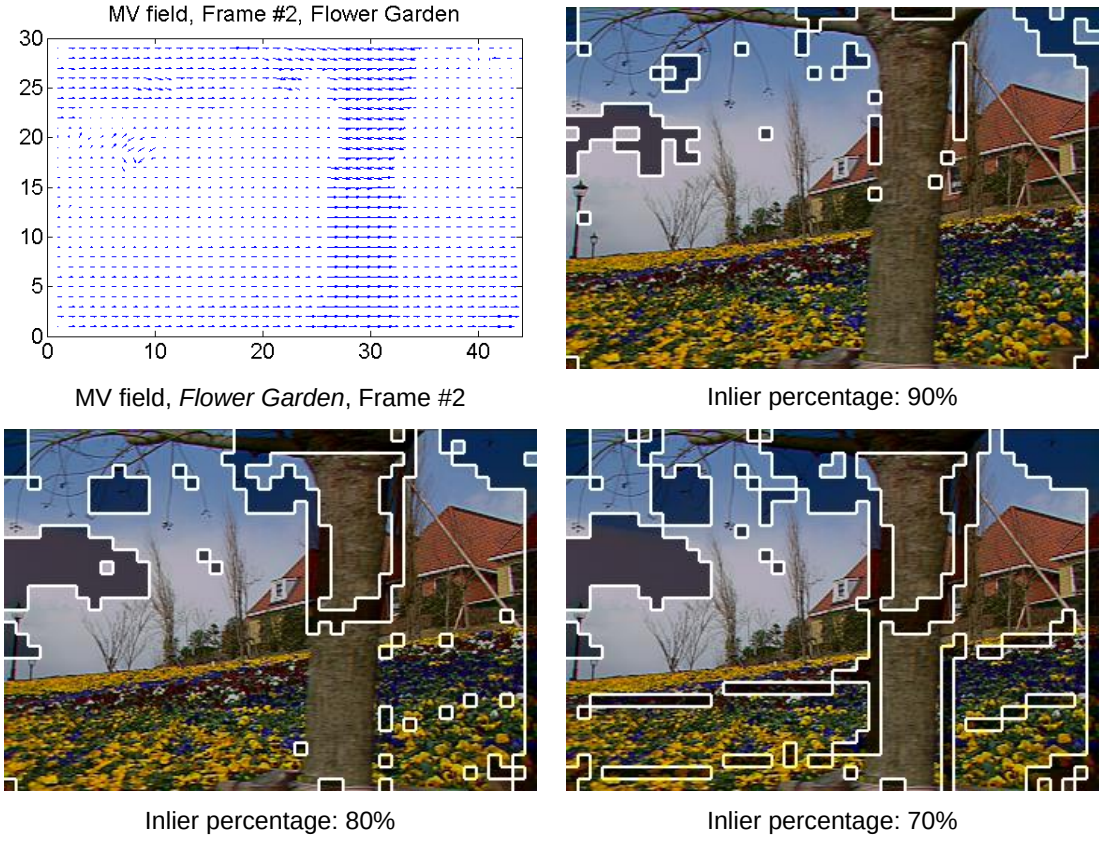


Figure 6.11: MV field from bitstream and outlier distribution with inlier percentage at 70%, 80% and 90%.

The outlier distributions show that the noisy MVs are mainly located in 1) areas with poor texture (e.g. background sky) and 2) boundary areas of moving foreground objects (e.g. the tree trunk). Also note outliers are seen along the frame boundaries. These outliers are mainly caused by camera movement. The areas are new appearances in a frame as the camera moves and generally end up with fairly different MVs compared to their neighbouring blocks, despite their rich texture. As we decrease the MV inlier percentage from 90% down to 70%, more MV outliers are discovered, but they mainly appear in these areas. An MV inlier percentage of 80% is empirically chosen in this section and used throughout the rest of the experiments.

We use the 8-parameter perspective model, described by a vector $\mathbf{m} = [m_0, \ldots, m_7]$, to represent GM. The parameters are estimated using an iterative gradient descent approach [3]. Once we obtain GM parameters global MVs can be calculated for each pixel in the current frame. Given (x, y) and (x', y') as the coordinates in the current and the previous frame, respectively, corresponding to the global motion model $\mathbf{m}$, the *X*- and *Y*- components of the global MV (GMV) at (x, y) in the current frame can be computed as:

$$GMV_{x,y}^{X}(t; \mathbf{m}) = x' - x,$$
$$GMV_{x,y}^{Y}(t; \mathbf{m}) = y' - y, \qquad \text{Equation 6.2}$$

where

$$x' = \frac{m_0 x + m_1 y + m_2}{m_6 x + m_7 y + 1},$$
$$y' = \frac{m_3 x + m_4 y + m_5}{m_6 x + m_7 y + 1}, \qquad \text{Equation 6.3}$$

For local motion we perform MV field refinement. The goal of MV field refinement is to find a reliable MV to replace the original MV that is identified as an outlier. Since block-matching-based motion estimation sometimes fails to catch a reliable MV a better approach is to find an MV replacement from its surrounding reliable MVs. The criterion is the minimum prediction error between the current block and the block from the reference frame. Suppose the MV of a $b \times b$ pixel block centered at (x, y) has been declared an outlier, and let $\mathbf{IMV}_n = (IMV_n^X, IMV_n^Y)$, $n = 0,1, \ldots, N-1$, be its N surrounding MV inliers. For each neighboring inlier MV we calculate the sum of absolute differences (SAD) between this block in the current frame $\mathbf{f}^{curr}$ and the corresponding block in the previous frame $\mathbf{f}^{prev}$:

$$D_{x,y}(\mathbf{IMV}_n) = \sum_{i=-\frac{b-1}{2}}^{\frac{b-1}{2}} \sum_{j=-\frac{b-1}{2}}^{\frac{b-1}{2}} \left(SAD_{x,y,i,j}(\mathbf{IMV}_n)\right) \quad \text{Equation 6.4}$$

$$SAD_{x,y,i,j}(\mathbf{IMV}_n) = |\mathbf{f}^{curr}(x+i, y+j) - \mathbf{f}^{prev}(x + IMV_n^X + i, y + IMV_n^Y + j)|, \quad \text{Equation 6.5}$$

The best MV is the one that minimizes $D_{x,y}$. This vector replaces the outlier MV as the new (local) MV for the block at (x, y):

$$\mathbf{MV}(x, y) \leftarrow \text{argmin}_{\mathbf{IMV_n}}\left(D_n(\mathbf{IMV}_n)\right) \quad \text{Equation 6.6}$$

By the end of this stage we obtain a global MV and local MV for each pixel in the current frame.

6.1.4.2. Pixel-level Global and Local MV Prediction

In the previous section, we obtained the global and refined local motion vectors that connect the current frame to the previous frame. The next step is to predict the motion vectors that connect the future (target) frame to the current frame. This way, each pixel in the target frame can be predicted using local and global motion, and these two predictions can be combined in a way that minimizes the overall prediction error.

Before we proceed on to local and global MV prediction, we first calculate the absolute inter-prediction residual $\mathbf{r}$ between previous frame $\mathbf{f}^{prev}$ and current frame $\mathbf{f}^{curr}$ using LMV and GMV from **Stage-1:**

$$\mathbf{r}(n, m) = \left|\mathbf{f}^{curr}(n, m) - \mathbf{f}^{prev}\left(n + MV_i^X, m + MV_i^Y\right)\right|, \quad \text{Equation 6.7}$$

where $n = 0, \cdots, N-1; m = 0, \cdots, M-1$, $M \times N$ is the frame resolution, and (MV_i^X, MV_i^Y) represents either **GMV** or **LMV** depending on whether prediction is done based on global or local motion.

GMV prediction for the future frame is similar to the calculation through Eq. 6.5 and 6.6. Assuming consistent camera motion between consecutive frames, we apply the same GM parameters from **Stage-1** for GMV prediction.

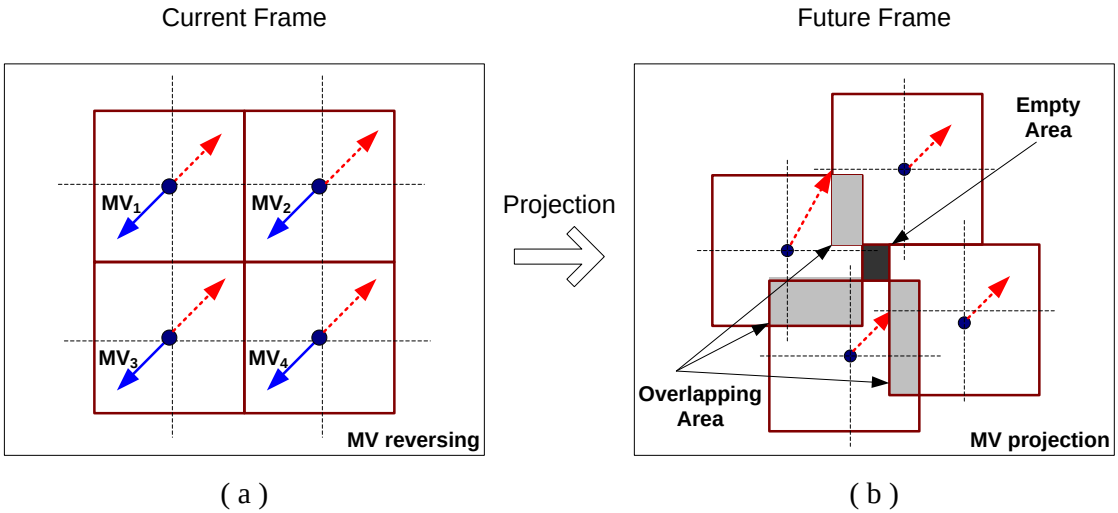


Figure 6.12: Local MV prediction, (a): MV reversing, (b): MV projection

LMV prediction, on the other hand, is accomplished through an iterative approach. First, we reverse refined local MVs from **Stage-1** and point them towards the future frame, as shown in Fig. 6.12(a). The difference between neighbouring MVs may lead to: 1) an overlapping area where pixels are landed on by multiple MVs, and 2) an empty area where pixels are not landed by any MVs, as shown in Fig. 6.12(b).

To solve case 1), i.e. an overlapping area, the pixel is assigned an MV that minimizes the inter-prediction residual among all candidate MVs . For the pixel located at (x, y) in the future frame that has N MVs pointing to it, $\mathbf{LMV}_i, i = 0,1, \ldots, N-1$, we pick the MV that leads to a minimum residual and declare it the final predicted LMV:

$$\mathbf{LMV}(x,y) \leftarrow \text{argmin}_{\mathbf{LMV}_i}\left(\mathbf{r}\left(x + LMV_i^X, y + LMV_i^Y\right)\right) \quad \text{Equation 6.8}$$

To solve case 2), i.e. an empty area, we apply the same approach, however, candidate LMVs are formed from surrounding pixels in a 3×3 neighbourhood. For a small empty area, whose either width or height is less than 3 pixels, we can always find LMV candidates to perform the prediction, while for a large empty area, i.e. both width and height are greater than 3 pixels, there will be some pixels surrounded by pixels with no LMVs. To deal with such a case, we use an iterative approach to gradually fill up the empty area. We start with pixels that are surrounded by at least one pixel with a valid LMV and predict MVs for those pixels using Eq. 6.8. Then, a newly filled area is used for the next round of LMVs prediction. Iteratively, we predict LMVs for all pixels in the future frame.

6.1.4.3. Frame Synthesis using LMMSE Criteria

With global and local MVs predicted from **Stage-2** we are able to temporally predict two future frames based on GMVs and LMVs. In this final stage we merge these two frames based on the Linear Minimum Mean Square Error (LMMSE) criterion [109], under the assumption of consistent global and local motion between consecutive frames. Please refer to Chapter 5, Section 5.3, for the derivation of LMMSE weights.

The key to LMMSE is to determine the error resulted from temporal prediction through either local or global motion. However, it is impossible to determine the actual error as we do not have the knowledge of the future frame (if we knew the future frame, there would be no need to predict it). Assuming consistent camera motion and a steady object moving trajectory, we use previous inter-prediction error statistics as an alternative, as shown in Fig. 6.13. With a global MV and a refined local MV from the current frame (frame n in the figure) we can "predict" the current frame from the previous frame and thus calculate two inter-prediction error frames, E_{GM} and E_{LM}, corresponding to prediction using global and local motion, respectively. We then use predicted GMV and LMV of the future frame to move prediction error residuals from E_{GM} and E_{LM} onto the future frame. This forms two predicted error frames $\mathbf{e}_{GM}$ and $\mathbf{e}_{LM}$ that serve as predicted error signals for the synthesis of the future frame via global and local motion,

respectively. We further partition $\mathbf{e}_{GM}$ and $\mathbf{e}_{LM}$ into 2×2 blocks. The error variance of each 2×2 block is used to calculate the weights w_{LM} and w_{GM} in Eq. 6.1, which are in turn applied to all four pixels inside the 2×2 block. The weights are calculated using the LMMSE approach described in Section 5.3.

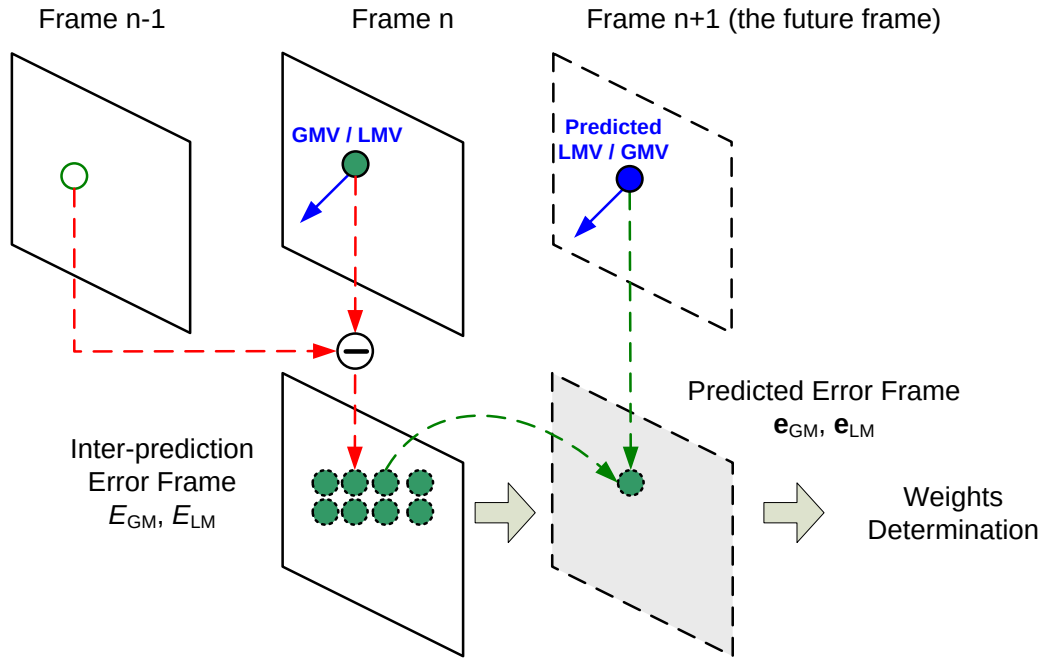


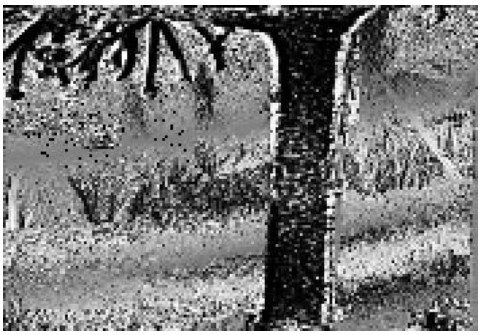
Figure 6.13: Prediction error and weights determination.

Predicted frame (GM)

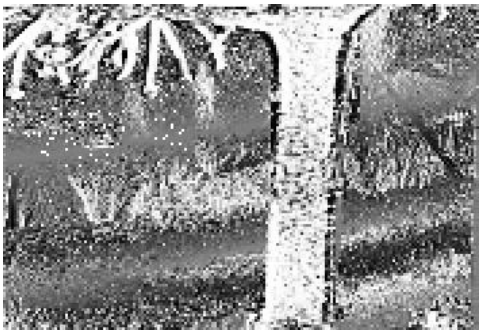
Predicted frame (LM)

Predicted residual (GM)

Predicted residual (LM)



LMMSE weights (GM)



LMMSE weights (LM)

Figure 6.14: [Top]: Predicted frame, [Middle]: prediction residual, [Bottom]: computed LMMSE weights. Frame prediction using global (left) and local motion (right).

In Fig. 6.14, using frame #2 of *Flower Garden*, we show frames predicted by local and global motion (top row), predicted error frames $\mathbf{e}_{GM}$ and $\mathbf{e}_{LM}$ (middle row), and

visualized weights w_{LM} and w_{GM} (bottom row). The results for global motion are shown in the left column and for local motion in the right column. Note in the residual frame, a brighter colour represents higher prediction error, while in the LMMSE weights frame, a brighter colour represents higher weights.

Though visually two predicted frames are hard to distinguish in the top row, the residual frames in the middle row illustrate that GM is better in predicting background areas (e.g. the garden and house), while LM performs better in predicting foreground objects (e.g. the tree trunk). The LMMSE weights reflect how GM and LM should influence the final target frame. From the right column of Fig. 6.14 we can see LM gets high weights on foreground areas and some parts of the background areas that have rich texture, while GM gets high weights in most of the background areas.

6.1.4.4. Subjective and objective quality evaluation

In this section we compare the subjective quality of predicted frames as well as their prediction error. The prediction is tested up to 3 frames ahead using the following three prediction methods:

- **Method-1** is the block-based predictive decoding based on whole frame concealment approach proposed in [88],
- **Method-2** is the region segmentation based approach from section 6.1.3, also refer to our earlier work [92],
- **Method-3** is the just described method based on LMMSE combination of local and global temporal prediction.

In Fig. 6.15 we display frames predicted from these three methods. In this experiment, frame #33 of *Coastguard* is predicted 3 frames ahead of time.

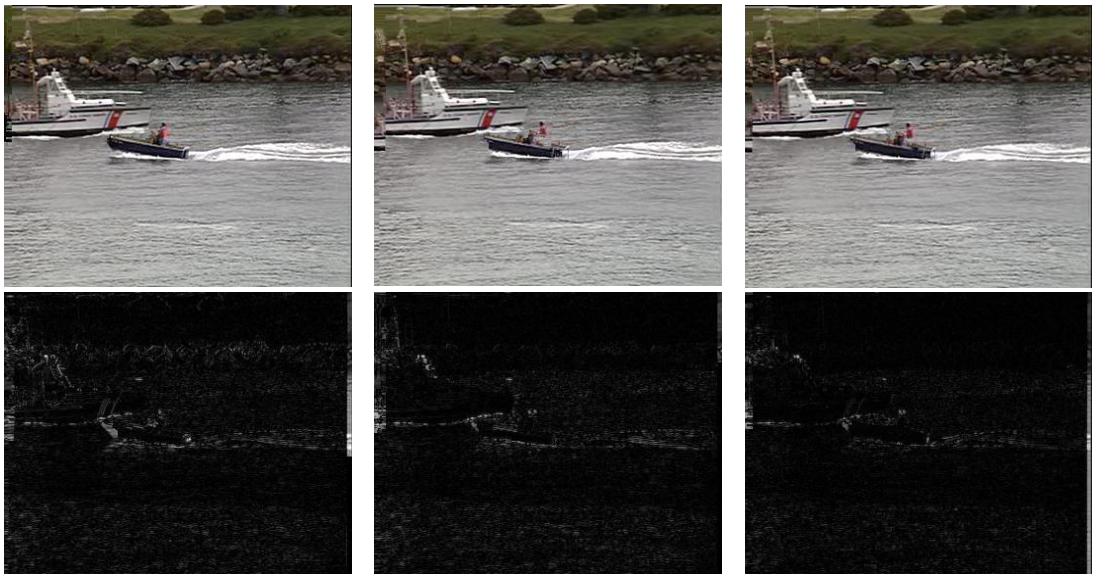


Figure 6.15: Predicted frame and their error frames, frame #33 of *Coastguard*. [Left]: Method-1, PSNR: 23.84dB, [Middle]: Method-2, PSNR: 24.75dB, and [Right]: Method-3, PSNR: 25.21dB.

As we predict a frame that is 3 frames ahead of the current frame we expect the artefacts to be more or less in all predicted frames, while error frames illustrate where the artefacts are located, e.g. object boundaries and textured areas. We observe that the latest method (**Method-3**) benefits from modeling global motion and results in less prediction error in the background area than **Method-1** and **Method-2**, while **Method-2** seems to have less background prediction error than **Method-1** due to its region-based approach. Also, despite a high prediction residual along the object boundaries for all three approaches, **Method-3** seems to have less overall prediction error on foreground objects, with 0.47dB PSNR gain over **Method-2** and 1.37dB over **Method-1**. This shows, at some level, that weighted local and global temporal prediction performs more effectively than the other two pure local motion based approaches.

Finally, we present objective evaluation results in terms of average PSNR for three approaches on 6 sequences. PSNR is computed with a prediction depth up to 3 frames ahead, as shown in Fig.16. We can observe that **Method-3** performs better overall than the two other approaches when the sequence contains global motion, such

as *Flower Garden*, *Tempete*, *Coastguard* and *City*. The PSNR gain of **Method-3** is about 1~2dB over **Method-2** and up to 4~5dB over **Method-1**. For sequences with a stationary camera, *Tennis* and *Mother & Daughter*, **Method-3** performs closely with **Method-2**. For sequences that involve very little motion, e.g. *Mother & Daughter*, all methods have a comparable (and very good) performance.

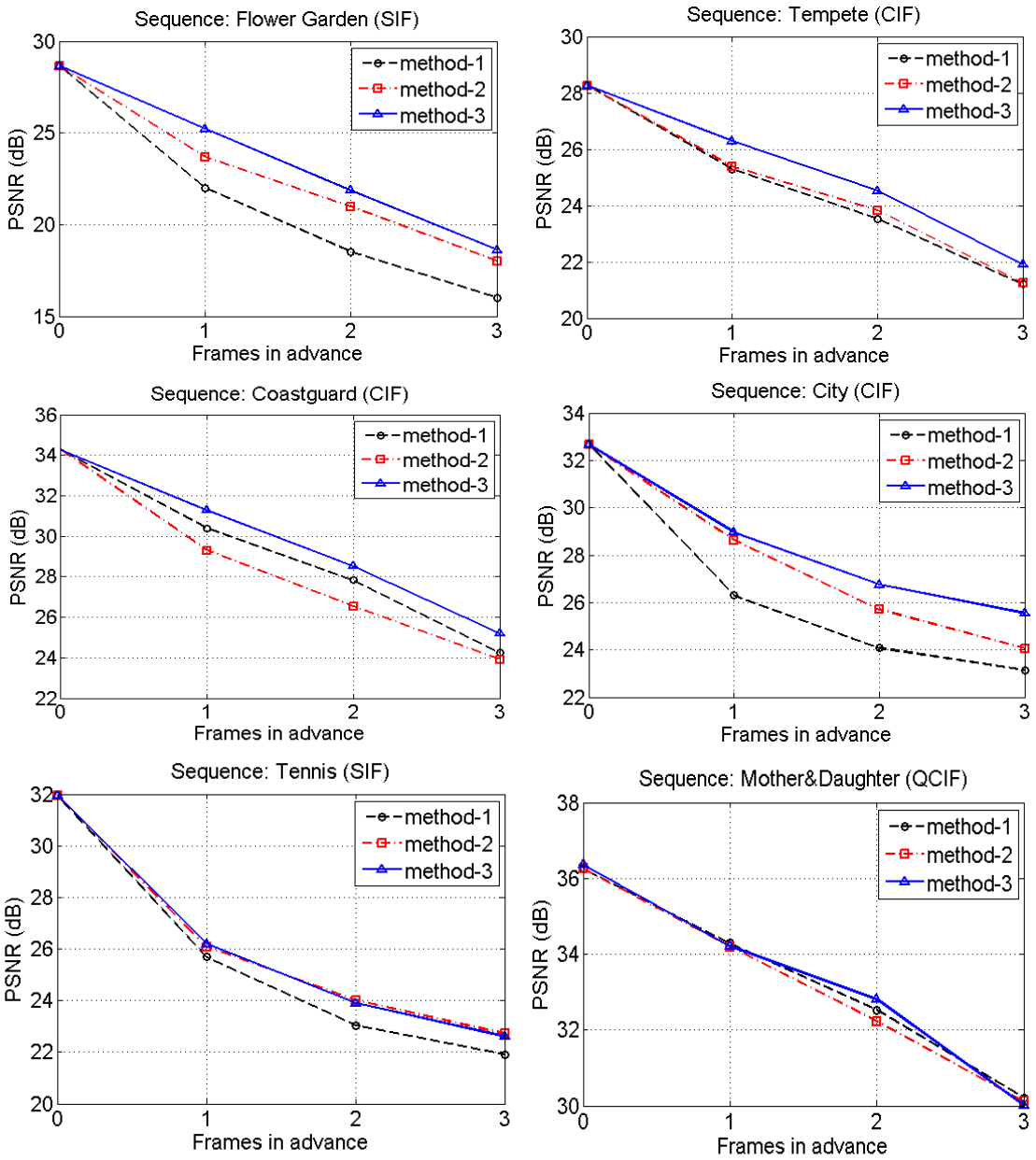


Figure 6.16: PSNR vs. prediction depth.

6.2. Frame Interpolation

A useful application with frame interpolation is the frame rate up-conversion (FRUC), which increases temporal resolution of video by interpolating frames between existing video frames. It has been proposed as a tool for video communications, where video sources sometimes have to be temporally downsampled to meet low bandwidth requirements of communication channels. FRUC also plays a role in other video applications, such as display format conversion and slow motion playback [110]. Various FRUC algorithms have been developed in the past. Non-motion compensated schemes, such as frame repetition and frame averaging, interpolate a skipped frame by combining the pixel values at the same location between neighbouring frames. These schemes are fast, but lead to blurring and motion jerkiness when strong motion is present in the scene.

To improve visual quality, a frame can be interpolated based upon on the motion vectors (MVs) between two consecutive frames, leading to motion-compensated frame interpolation (MCFI) [95] [110-112]. Due to the low density of block-based MVs and their inaccuracies, direct MCFI causes annoying visual artefacts in interpolated frames. Many approaches have been proposed to compensate for these artefacts, including vector median filtering, overlapped block motion compensated (OBMC) interpolation [110], multistage refinement [111-112], bilateral [95] and global motion estimation [113].

Alternatively, instead of using block-based motion, region-based schemes interpolate the frames by considering moving regions or objects. In [114], a region-based interpolation method is proposed, which characterized the region motion using an affine 4-parameter motion model and determined depth order based on statistical analysis of occluded areas. This approach is effective in removing visual artefacts and preserving object structure, but it is computationally fairly expensive.

In this section, we present a region-based frame interpolation (RBFI) scheme for compressed video. The computational complexity is reduced by utilizing MVs from the compressed bitstream instead of re-estimating them. Similarly to [13], we incorporate a scale parameter into our motion model and estimate region depth order by analyzing the motion of neighbouring regions.

6.2.1. *Interpolation Framework*

RBFI takes segmented moving regions as basic units to generate the interpolated frame. In Fig. 6.17, frame t is first segmented into moving regions and these regions are tracked into frame $t+1$. In this example, there are three regions (R_1, R_2, R_3) after segmentation: two moving regions (R_1 and R_2) and one background region (R_3). Region depth order is then estimated as explained in Section 5.2.1. Assuming R_1 is the closest to the point of view, and R_3 is the furthest, we should obtain depth order ideally as $D_3 > D_2 > D_1$. The interpolated frame $\mathbf{f}_{t+m}$ is generated by:

$$\mathbf{f}_{t+m} = w_f \cdot \mathbf{f}_{t\to t+m} + w_b \cdot \mathbf{f}_{t+1\to t+m} \qquad \text{Equation 6.9}$$

$$\mathbf{f}_{t\to t+m} = R_1^{t\to t+m} \oplus R_2^{t\to t+m} \oplus R_3^{t\to t+m}$$
$$\mathbf{f}_{t+1\to t+m} = R_1^{t+1\to t+m} \oplus R_2^{t+1\to t+m} \oplus R_3^{t+1\to t+m} \qquad \text{Equation 6.10}$$

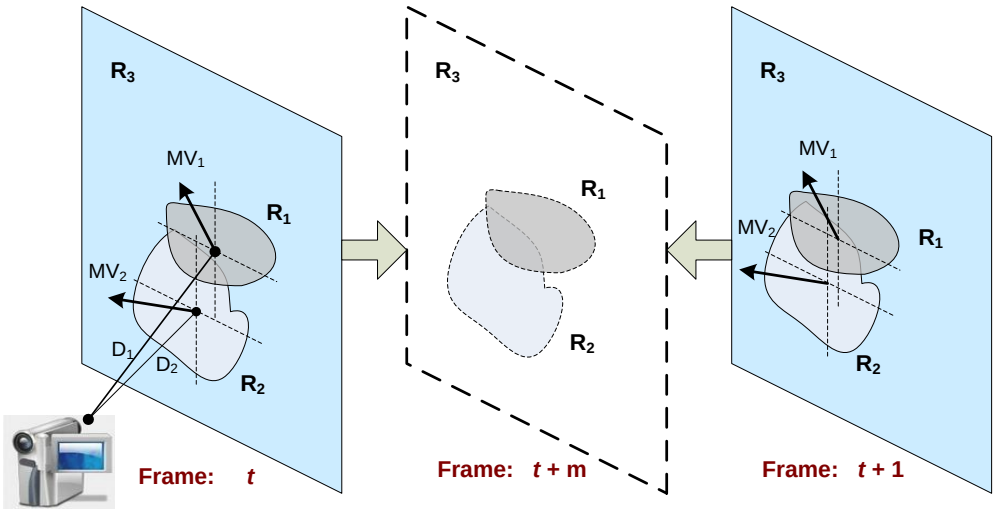


Figure 6.17: **Region-based frame interpolation (RBFI).**

where $m \in (0,1)$ denotes the relative position of the interpolated frame (e.g. $m = 0.5$ for 1:2 up-conversion) and (w_f, w_b) are the weights for the forward and backward

interpolation, respectively. We simply used $(w_f, w_b) = (0.5, 0.5)$ in our experiments, because we only considered 1:2 up-conversion.

Equation 6.9 is used to generate forward and backward interpolated frames $\mathbf{f}_{t\to t+m}$ and $\mathbf{f}_{t+1\to t+m}$. Here, $R_k^{t\to t+m}$represents the displacement of region R_k from the image plane at time t into the image plane at time $t+m$ along its predicted trajectory. The operation $\oplus$ represents the aggregation of pixels (x, y) from different regions, defined as follows:

$$R_i \oplus R_j(x,y) = \begin{cases} R_i(x,y), & (x,y) \in R_i \backslash R_j \\ R_j(x,y), & (x,y) \in R_j \backslash R_i \\ R_k(x,y), & (x,y) \in R_i \cap R_j \end{cases} \qquad \text{Equation 6.11}$$

where $k = \arg\min_{k\in\{i,j\}}(D_i, D_j)$. Region depths D_i and D_j determine which region gets occluded during frame synthesis. In Fig. 6.17, region R_1 will be "in front of" R_2 and R_3, while region R_3 will be "behind" R_1 and R_2.

6.2.2. Moving Region Segmentation and Depth Order Estimation

Motion information exists in a compressed video stream as MVs, which are associated with blocks of various sizes (e.g. 8×8 and 16×16 in MPEG-4, 4×4 to 16×16 in H.264/AVC). We first normalize MVs according to the temporal distance and direction indicated by the reference frame index and then map them to the minimum block size supported by the particular codec.

In our earlier work [27] we proposed a coarse-to-fine algorithm for moving region segmentation. The same strategy, shown in Fig. 6.18, is adopted here. It consists of two stages: 1) coarse segmentation, which gives regions with block-based boundaries obtained either by motion segmentation or region tracking, and 2) fine segmentation using colour and edge information, which refines region boundaries to pixel precision.

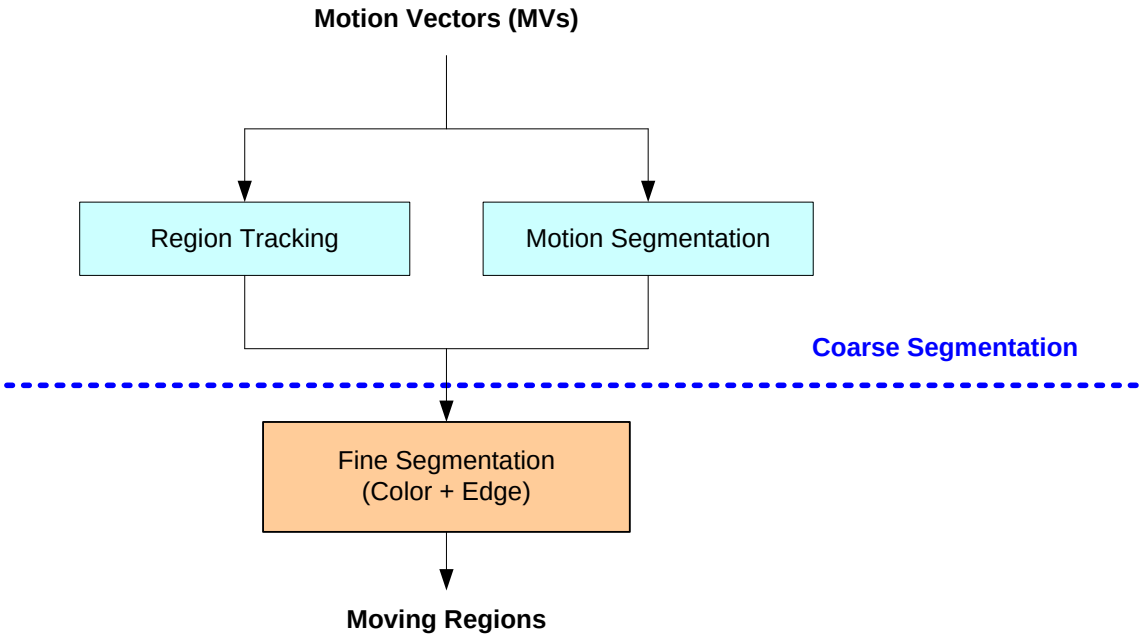


Figure 6.18: **Coarse-to-fine moving region segmentation.**

Let frame t and $t+1$ be the two frames between which we wish to interpolate. Motion segmentation is carried out in frame t by our algorithm from [27], while in frame $t+1$, coarse segmentation is performed with the help of region tracking, illustrated in Fig. 6.19. Blocks in frame $t+1$ are grouped by examining their reference blocks in frame t. If the reference block in frame t is located inside of some region the current block inherits the same region index, otherwise, the current block is declared as a part of the boundary region and is subject to further boundary refinement. In this way, we can maintain the consistency of moving regions from one frame to the next.

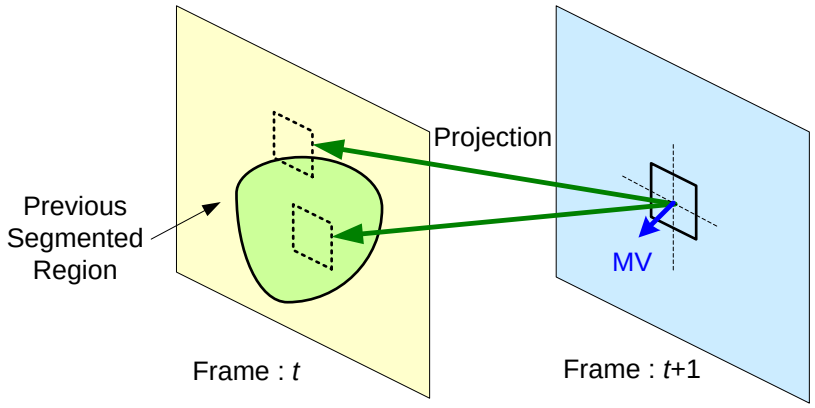


Figure 6.19: **Coarse segmentation by region tracking.**

The maximum number of moving regions is set to 15 in our experiments. If the number of regions after motion segmentation is greater than 15, iterative region merging is performed to reduce their number to 15. Then, the depth of each region is estimated as discussed in Chapter 5, Section 5.2.

Motion segmentation described above identifies moving regions in a frame. A crucial step of synthesizing an intermediate frame is to predict the moving trajectory, i.e. how these moving regions move backward or forward to their neighbouring frames. We approximate the trajectory of a region using a 3-parameter motion similar to the model used in [13].

Let S^R be the scale factor, v_{hori} and v_{vert} be the horizontal and vertical velocity, respectively, and (x, y) be the coordinates of a pixel in region R in frame t. Then the corresponding coordinate of this pixel in frame $t+1$ can be computed by:

$$\begin{pmatrix} x \\ y \end{pmatrix}_{t+1}^{R} = S^R \cdot \left[\begin{pmatrix} x \\ y \end{pmatrix}_{t}^{R} + \begin{pmatrix} v_{hori} \\ v_{vert} \end{pmatrix}^{R} \right] \qquad \text{Equation 6.12}$$

If the up-conversion factor is $1{:}\,M$, then at time $(t - m/M)$ and $(t + m/M)$, where $m = 1,2,\ldots,M-1$, the pixels of region R are at

$$\binom{x}{y}^{R}_{t-\frac{m}{M}} = \frac{M-m}{M} \cdot \left[\left(1+\frac{1}{S^R}\right) \cdot \binom{x}{y}^{R}_{t+1} - \binom{v_{hori}}{v_{vert}}^{R}\right] \quad \text{Equation 6.13}$$

$$\binom{x}{y}^{R}_{t+\frac{m}{M}} = \frac{m}{M} \cdot \left[(1+S^R) \cdot \binom{x}{y}^{R}_{t} + S^R \cdot \binom{v_{hori}}{v_{vert}}^{R}\right] \quad \text{Equation 6.14}$$

This motion model is based on the global camera motion model proposed in [13]. By applying the same motion parameters to all pixels in a region we can keep that region intact during the synthesis of an intermediate frame and suppress, to some extent, blocking artefacts found in block-based interpolation.

For the k-th block of a region in frame t, let $\mathbf{c'}_k = (x'_k, y'_k)^T$ denote the coordinates of the block center with respect to the region center. Then the corresponding coordinate of the block center in frame $t-1$ can be obtained as:

$$\mathbf{c}_k = \mathbf{c'}_k + \mathbf{MV}_k = \binom{x'_k}{y'_k} + \binom{MV_x}{MV_y} \quad \text{Equation 6.15}$$

The three motion parameters, scale factor, and horizontal and vertical velocity can be estimated as follows [13]:

$$S^R = \frac{\sum_{k=1}^{N}\left((\mathbf{c'}_k - \bar{\mathbf{c}}')^T(\mathbf{c}_k - \bar{\mathbf{c}})\right)}{\sum_{k=1}^{N}\|\mathbf{c}_k - \bar{\mathbf{c}}\|^2} \quad \text{Equation 6.16}$$

$$\binom{v_{hori}}{v_{vert}} = \frac{\bar{\mathbf{c}}'}{S^R} - \bar{\mathbf{c}} \quad \text{Equation 6.17}$$

$$\bar{\mathbf{c}}' = \frac{1}{N}\sum_{k=1}^{N}\mathbf{c}'_k \quad \text{Equation 6.18}$$

$$\bar{\mathbf{c}} = \frac{1}{N}\sum_{k=1}^{N} \mathbf{c}_k$$

where N is the total number of blocks in the region.

Encoded MVs are commonly estimated by minimizing the prediction error at the encoder, so it is often found that some MVs do not match the physical region/object motion. In [13], these unreliable MVs are iteratively removed from the motion vector field by several heuristics. We instead apply our outlier removal algorithm [23] to obtain a more reliable MV field prior to motion parameter estimation in Eq. 6.16-6.18.

6.2.3. Implementation of RBFI in MPEG-4

Figure 6.20 shows the functional diagram of RBFI within a block-based video decoder, in this case MPEG-4. The key information fed into the frame interpolation model is two reconstructed frames (frames t and $t+1$) and their MVs. The framework shown in Fig. 6.20 operates on compressed video, but it can be adapted to raw video by replacing the video decoder with a motion estimator, where an extra frame $t-1$ is needed for motion estimation.

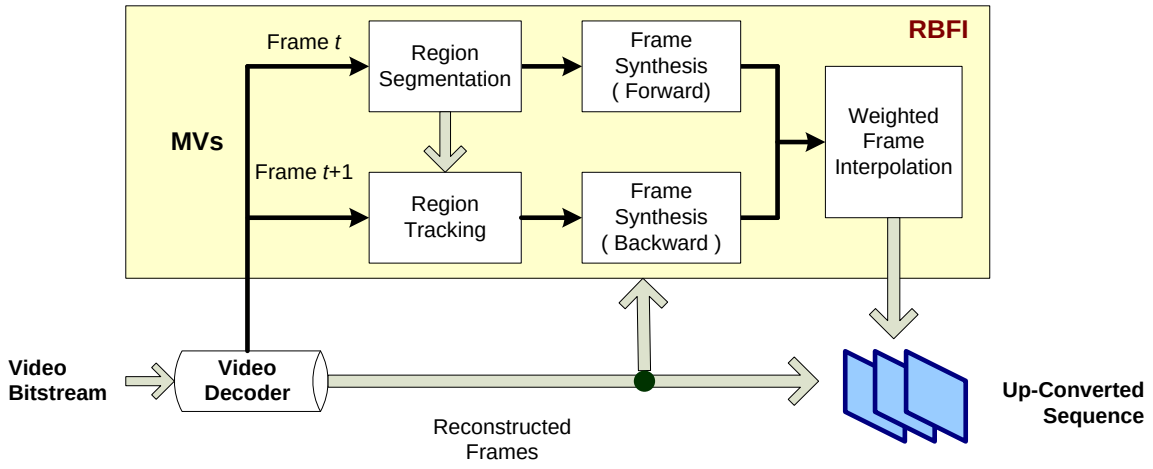


Figure 6.20: **Functional block diagram of the RBFI system.**

The popular Xvid MPEG-4 video codec (www.xvid.org) is used in our work to illustrate the proposed frame rate up-conversion. The steps are:

Step 1. Segmenting moving regions in frame t and tracking all regions towards frame $t+1$;

Step 2. Estimating three motion parameters of all moving regions in frames t and $t+1$, and averaging these two sets of motion parameters to predict final motion parameters for a particular region;

Step 3. Determining region depth order using predicted region moving trajectories, as in Section 5.2;

Step 4. Forward (backward) synthesis of an intermediate frame $\mathbf{f}_{t\to t+m}$ ($\mathbf{f}_{t+1\to t+m}$) using decoded frame t ($t+1$);

Step 5. Post-processing two synthesized frames:

a. Overlapped areas: determined using the estimated region depth order;

b. Empty areas: a combination of boundary matching and linear spatial interpolation [52];

Step 6. Interpolating frame $\mathbf{f}_{t+m}$ by weighted averaging of two intermediate frames using Eq. 6.8.

6.2.4. Experimental results

For performance evaluation purpose, five video sequences — *Foreman*, *Football*, *Soccer*, *Mobile Calendar* and *Crew* — of CIF resolution (352×288) at 30 frames per second (fps) are used in the experiments. The frame rate is first reduced to 15 fps for all sequences by skipping even frames and the Xvid MPEG-4 video encoder is then used to compress sequences with a bitrate set to 512 kbps using the IPPPP GOP structure. With these testing sequences, we compare the frame interpolation performance of the following four methods:

- **Method-1** – frame repetition;

- **Method-2** – frame averaging;
- **Method-3** – block-based MCFI [110];
- **Method-4** – the proposed region-based frame interpolation.

The visual comparison of interpolated frames is presented in Figs. 6.21–23 for sequences *Foreman*, *Football* and *Soccer*, respectively. In Fig. 6.21, blurring caused by **Method-2** is seen in the top right frame. **Method-3** solves this problem by using motion compensation, but introduces blockiness in the bottom left frame. The block artefacts are mostly removed by **Method-4** in the bottom right.

Figure 6.22 shows an example from sequence *Soccer*. In the top right frame (**Method-2**) we can clearly see the double player caused by direct frame averaging. Although the bottom left frame (**Method-3**) is clearer than the frame produced by **Method-2**, part of it (players' hands and legs) is degraded due to motion inaccuracy. These artefacts have successfully been eliminated using **Method-4** in the bottom right of Fig. 6.22.

Similar results can also be observed in Fig. 6.23, where an example from *Football* is shown. Fast motion is involved as players are engaged and **Method-2** produces a blurred result. **Method-3** creates blocking artefacts in high-motion areas. **Method-4** provides much better visual quality than either **Method-2** or **Method-3**. Note, however, that despite their better visual quality the PSNR of frames produced by **Method-4** is not always higher than that of **Method-3**.

Figure 6.21: Interpolated frame #140 of *Foreman*. [Top Left], the original frame, [Top right]: Method-2 (PSNR: 27.43dB), [Bottom Left]: Method-3 (PSNR: 29.59dB), [Bottom Right]: Method-4 (PSNR: 29.42dB).

Figure 6.22: Interpolated frame #28 of *Soccer*. [Top Left], the original frame, [Top right]: Method-2 (PSNR: 23.28 dB), [Bottom Left]: Method-3 (PSNR: 25.91 dB), [Bottom Right]: Method-4 (PSNR: 26.16 dB).

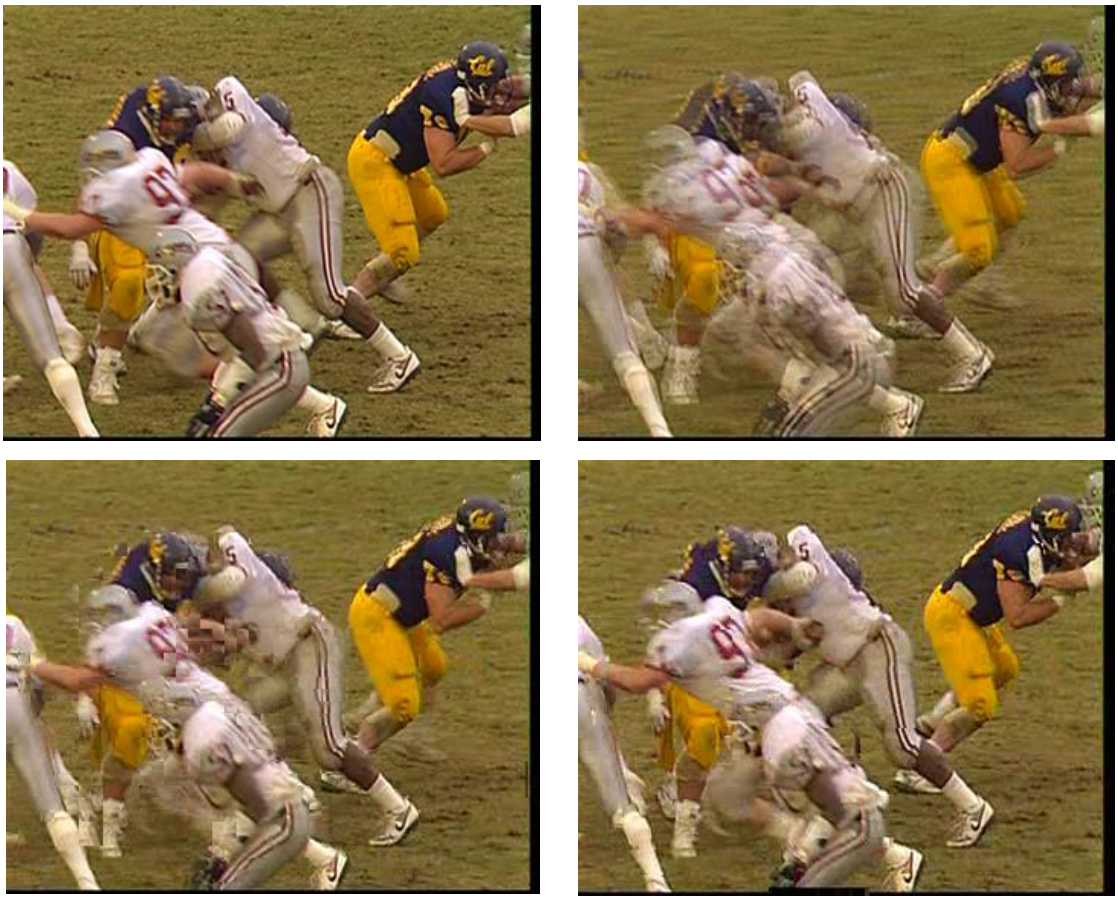


Figure 6.23: Interpolated frame #22 of *Football*. [Top Left], the original frame, [Top right]: Method-2 (PSNR: 25.46 dB), [Bottom Left]: Method-3 (PSNR: 27.22 dB), [Bottom Right]: Method-4 (PSNR: 27.13 dB).

We finally evaluate the proposed method in terms of the Peak Signal-to-Noise Ratio (PSNR) in dB. We list averaged PSNR values for these five video sequences in Table 6.2. As one might notice, the PSNR performance of the proposed **Method-4** is not always better than **Method-3**, even though our method generally has better visual quality than other methods. This is because PSNR measures pixel-wise prediction accuracy to its original frame, instead of perceived visual quality. The proposed region-based method maintains a good visual quality by partitioning the scene into a set of moving regions, and modelling the motion of the entire regions, rather than individual blocks. Although this gives visually pleasing results the fact that the region motion model

has only three parameters inevitably leads to some loss of accuracy, as evidenced by the PSNR results.

Table 6.2: Average PSNR comparison among four methods

Sequence	Method-1	Method-2	Method-3	Method-4
Foreman	26.33	28.32	30.52	**29.66**
Crew	25.89	27.85	30.49	**30.05**
Football	25.70	26.64	28.05	**27.84**
Soccer	20.96	23.33	26.34	**26.51**
Mobile Calendar	21.75	23.81	25.52	**25.93**

6.3. Frame Resizing

In this section, we address one of popular applications of frame resizing, Super-Resolution (SR) [105] [106], using the proposed techniques of motion modelling and object segmentation. SR is a technique for enhancing the resolution of images and video frames and plays an important role in a variety of applications, including video conferencing, video streaming and surveillance. Due to the abundance of compressed video content, SR reconstruction from compressed video has become important.

The utility of (sub-pixel) motion in resolution enhancement has long been recognized [107]. Using the motion vectors from the compressed stream, SR frames can be reconstructed via temporal interpolation, as in [108]. However, unreliable MVs may cause inaccurate interpolation, often evidenced by the occurrence of blocking artefacts in the SR frame. This problem can be mitigated by removing MV outliers [28] and/or refining MVs based on their neighbours [111]. Still, these techniques do not fundamentally address the problem of inaccuracy of local motion (i.e. MVs). Meanwhile, purely spatial approaches, e.g. bi-linear and B-spline interpolation [119], are free of motion artefacts, but may instead blur out textured regions in SR frames.

In this section, we propose a spatio-temporal SR approach for compressed video that incorporates both local and global motion modelling. The approach was present in our previous work [101]. The proposed approach makes use of the information from the

compressed bitstream, such as MVs and prediction residuals, and avoids their re-computation at the decoder. However, not all processing is done in the compressed domain; hence we do not call this approach "compressed-domain" SR, but simply SR from compressed video. The main idea is to reconstruct the SR frame through a fusion of temporally and spatially interpolated frames. The fusion weights are derived by assessing the accuracy of each interpolation method on a block-by-block basis and then combining them in a Linear Minimum Mean Squared Error (LMMSE) fashion. As shown in Fig. 6.24, we first extract MVs and inter-prediction residuals from the compressed video bitstream along with the low resolution (LR) frame(s). Following that, we form three preliminary versions of the SR frame: one based on temporal interpolation using local motion (LM), i.e. MVs; one based on temporal interpolation using global motion (GM); and one based on spatial interpolation. Based on these preliminary SR frames, a set of weights $\{w_S, w_{TL}, w_{TG}\}$ is determined for each block in the SR frame. The weights reflect the accuracy of each interpolation method in a particular block; the method with the highest accuracy will end up with the highest weight in that block. The final SR frame is obtained as a linear combination of the three preliminary SR frames on a block-by-block basis.

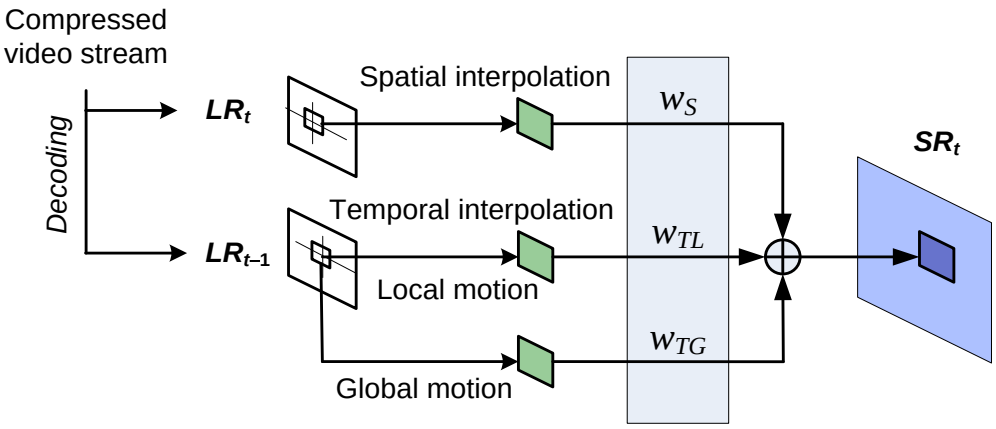


Figure 6.24: The proposed SR algorithm.

6.3.1. Spatio-Temporal SR Reconstruction

6.3.1.1. Spatial and Temporal SR reconstruction

In this thesis, spatial SR reconstruction refers to using the current low-resolution (LR) frame to interpolate the current SR frame, while temporal SR reconstruction refers to using the previous LR frame to interpolate the current SR frame. These two approaches can be linearly combined as follows:

$$\mathbf{f}_{SR}(x,y,t) = w_1 T_1\left(\mathbf{f}_{LR}\left(\frac{x}{s}+MV^X,\frac{y}{s}+MV^Y,t-1\right)\right) + w_2 T_2\left(\mathbf{f}_{LR}\left(\frac{x}{s},\frac{y}{s},t\right)\right) \quad \text{Equation 6.19}$$

where $\mathbf{f}_{LR}(x/s,y/s,t-1)$ and $\mathbf{f}_{LR}(x/s,y/s,t)$ denote LR frames at times $(t-1)$ and t, $\mathbf{f}_{SR}(x,y,t)$ denotes the SR frame at time t, MV^X and MV^Y are, respectively, the *X*- and *Y*- components of the MV at $(x/s,y/s)$ and T_1 and T_2 are the interpolators used for the previous and current LR frames, respectively. Pixel coordinates (x,y) refer to the SR frame and s is the scaling factor between the LR and SR frames, here assumed to be the same for both width and height. Weights w_1 and w_2 determine the influence of temporal and spatial interpolation on the resulting SR frame.

Figure 6.25 shows two extreme cases — purely spatial interpolation ($w_1 = 0, w_2 = 1$) on the left and purely temporal interpolation ($w_1 = 1, w_2 = 0$) on the right — on a frame from *Flower Garden* encoded using the Xvid MPEG-4 encoder with half-pixel accurate MVs at 512 Kbps. Note that the temporally reconstructed frame (right) preserves texture better than the one on the left, but suffers from motion artefacts near the boundaries of the tree trunk. Hence, different interpolators seem to be appropriate in different parts of the frame. In Chapter 5, Section 5.3, we developed a method to decide how much weight to assign to each interpolator using a Linear Minimum Mean Square Error (LMMSE) approach.

$w_1 = 0, w_2 = 1$ $w_1 = 1, w_2 = 0$

Figure 6.25: SR frame reconstruction using different weights on spatial and temporal interpolated pixels (*Flower Garden* frame #2).

6.3.1.2. SR Reconstruction Incorporating Local and Global Motion

Motion vectors (MVs) in the compressed video stream represent local translational motion estimates of individual blocks in the frame. Ideally, for accurate temporal SR reconstruction, we would like to have precise motion information for each pixel. In certain cases, especially in the background areas, a more accurate estimate of the motion of individual pixels may be obtained by utilizing global motion. Our final reconstructed SR frame, $\mathbf{f}_{SR}(x, y, t)$, is obtained as a linear combination of three preliminary SR frames: a temporal frame obtained on the basis of local motion (TL), i.e. MVs from the compressed bitstream; a temporal frame obtained on the basis of global motion (TG); and a spatial frame obtained through spatial interpolation. The SR reconstruction process can be represented as:

$$\mathbf{f}_{SR}(x,y,t) = w_{TL}T_1\left(\mathbf{f}_{LR}\left(\frac{x}{s}+MV^X,\frac{y}{s}+MV^Y,t-1\right)\right) + w_{TG}T_2\left(\mathbf{f}_{LR}\left(\frac{x}{s}+GMV^X,\frac{y}{s}+GMV^Y,t-1\right)\right) + w_S T_3\left(\mathbf{f}_{LR}\left(\frac{x}{s},\frac{y}{s},t\right)\right)$$

Equation 6.20

where (GMV^X, GMV^Y) is the global motion vector (GMV) corresponding to the location (x,y) in the SR frame. In this section, we use the 8-parameter perspective model described in the previous chapters, to represent GM. The parameter vector $\mathbf{m}$ is estimated in each frame from the MVs in the bitstream using an iterative gradient descent approach [3] with MV outlier rejection from [28]. Given $(x/s, y/s)$ and $(x'/s, y'/s)$ as the coordinates in the current and the previous LR frame, respectively, the *X*- and *Y*-components of the GMV at $(x/s, y/s)$ in the current LR frame corresponding to this motion model can be computed as:

$$GMV^X\left({}^{x}/_{s}, {}^{y}/_{s}; \mathbf{m}\right) = {}^{x'}/_{s} - {}^{x}/_{s},$$
$$GMV^Y\left({}^{x}/_{s}, {}^{y}/_{s}; \mathbf{m}\right) = {}^{y'}/_{s} - {}^{y}/_{s},$$

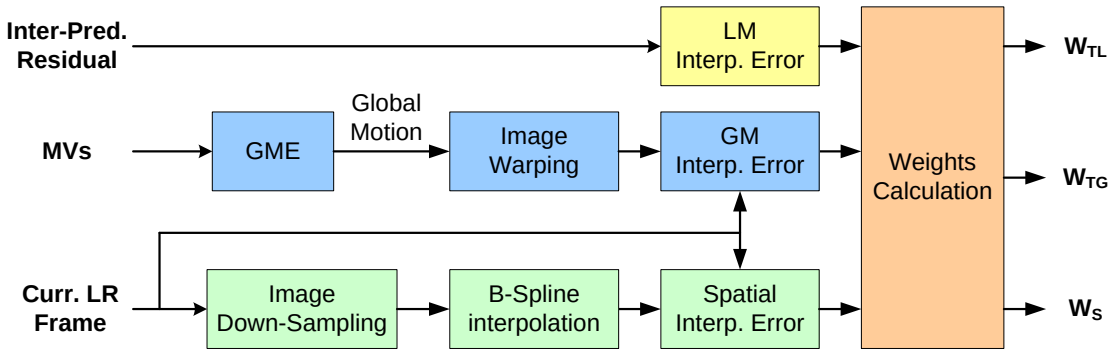
Equation 6.21

where

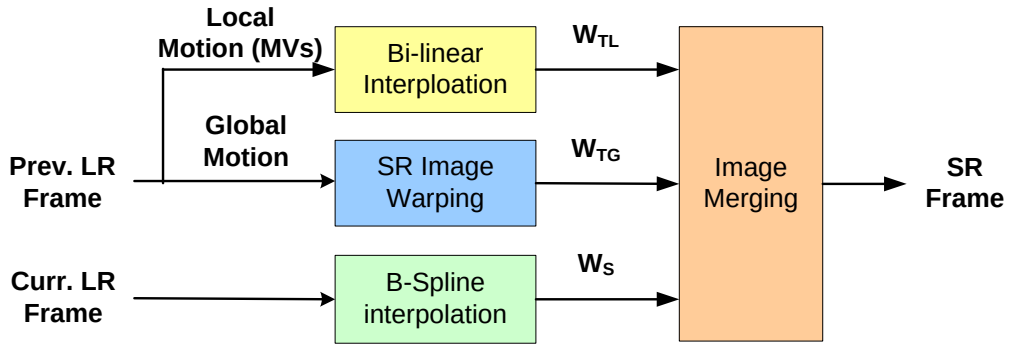
$${}^{x'}/_{s} = \frac{m_0\,{}^{x}/_{s} + m_1\,{}^{y}/_{s} + m_2}{m_6\,{}^{x}/_{s} + m_7\,{}^{y}/_{s} + 1}$$
$${}^{y'}/_{s} = \frac{m_3\,{}^{x}/_{s} + m_4\,{}^{y}/_{s} + m_5}{m_6\,{}^{x}/_{s} + m_7\,{}^{y}/_{s} + 1}$$

Equation 6.22

and $(x'/s, y'/s)$ is the location in the previous LR frame.



(a)



(b)

Figure 6.26: (a) LMMSE weight calculation, (b) SR frame reconstruction.

6.3.1.3. Combining Spatial and Temporal Frames

The weights on temporal and spatial interpolation should reflect the level of quality impact on the SR frame and are determined by the Linear Minimum Mean Square Error (LMMSE) criterion, which was discussed in Chapter 5, Section 5.3.

As discussed in Section 5.3, the LMMSE weights for each block depend on the interpolation error vectors $\{\mathbf{e}_{TL}, \mathbf{e}_{TG}, \mathbf{e}_S\}$ between the actual and interpolated SR blocks. Since the actual SR blocks are not known, these error vectors need to be estimated for each block. Thus, the proposed SR reconstruction proceeds as follows. First, we

estimate the interpolation error vectors $\{\mathbf{e}_{TL}, \mathbf{e}_{TG}, \mathbf{e}_S\}$ as shown in Fig. 6.26(a) for all blocks in the LR frame. In this section, we use 4×4 blocks, which, based on our experiments, offer a good balance between the accuracy of interpolation error estimates (which favours larger blocks) and visual quality of reconstructed frames (which favours smaller blocks to reduce blocking artefacts). The LM interpolation error $\mathbf{e}_{TL}$ is taken to be the decoded inter-prediction residual from the compressed bitstream. Hence, except for decoding, no other computation is required to obtain $\mathbf{e}_{TL}$. The GM interpolation error $\mathbf{e}_{TG}$ is obtained by first estimating the GM parameters of perspective motion using the methods from [28] and [3], and then warping the previous LR frame onto the current LR frame. The difference between the interpolated and the decoded current LR frame provides an estimate of $\mathbf{e}_{TG}$.

Finally, to estimate $\mathbf{e}_S$ we proceed as follows. We first down-sample the current LR frame by the same ratio that is to be used for LR → SR up-sampling (in this section all experimental results are based on scaling ratio $s = 2$). Then, we reconstruct the current LR frame from its down-sampled version using B-spline interpolation. The difference between this spatially-interpolated LR fame and the actual decoded LR frame provides an estimate of $\mathbf{e}_S$. Having obtained $\{\mathbf{e}_{TL}, \mathbf{e}_{TG}, \mathbf{e}_S\}$ as described above, we compute the LMMSE weights $\{w_{TL}, w_{TG}, w_S\}$ as discussed in Section 5.3.

Now that the LMMSE weights are computed, we proceed to synthesize the SR frame as shown in Fig. 5.26(b). First, preliminary versions of each 8×8 SR block are synthesized using LM, GM and spatial interpolation. The LM-interpolated block, $\hat{\mathbf{f}}_{TL}$, is generated using the decoded MVs together with bilinear interpolation, much like the conventional temporal SR reconstruction. The GM-interpolated block, $\hat{\mathbf{f}}_{TG}$, is generated using the estimated GM parameters through SR warping described by equations 6.20-6.22. The spatially-interpolated block, $\hat{\mathbf{f}}_S$, is generated using the B-spline interpolated block from the current LR frame. The final SR block is obtained as a linear combination of LM-, GM- and spatially-interpolated blocks.

6.3.2. Results and Discussion

In this section, we present experimental results to evaluate the performance of the proposed SR method. We compare our method with temporal and spatial SR

interpolation approaches from [118] and [119] on several standard test sequences (*Flower Garden* (QSIF), *Coastguard* (QCIF), *City* (CIF) and *Mobile Calendar* (QCIF)), all involving some amount of global motion. All sequences were in YUV 4:2:0 format and had a frame rate of 30 frames per second (fps). QSIF (176×120) and QCIF (176×144) resolution sequences were encoded at 128 kbps, while the CIF (352×288) sequence was encoded at 512 kbps. Encoding was performed using the Xvid MPEG-4 encoder. In all experiments the SR up-conversion factor was $s = 2$ in both the horizontal and vertical direction.

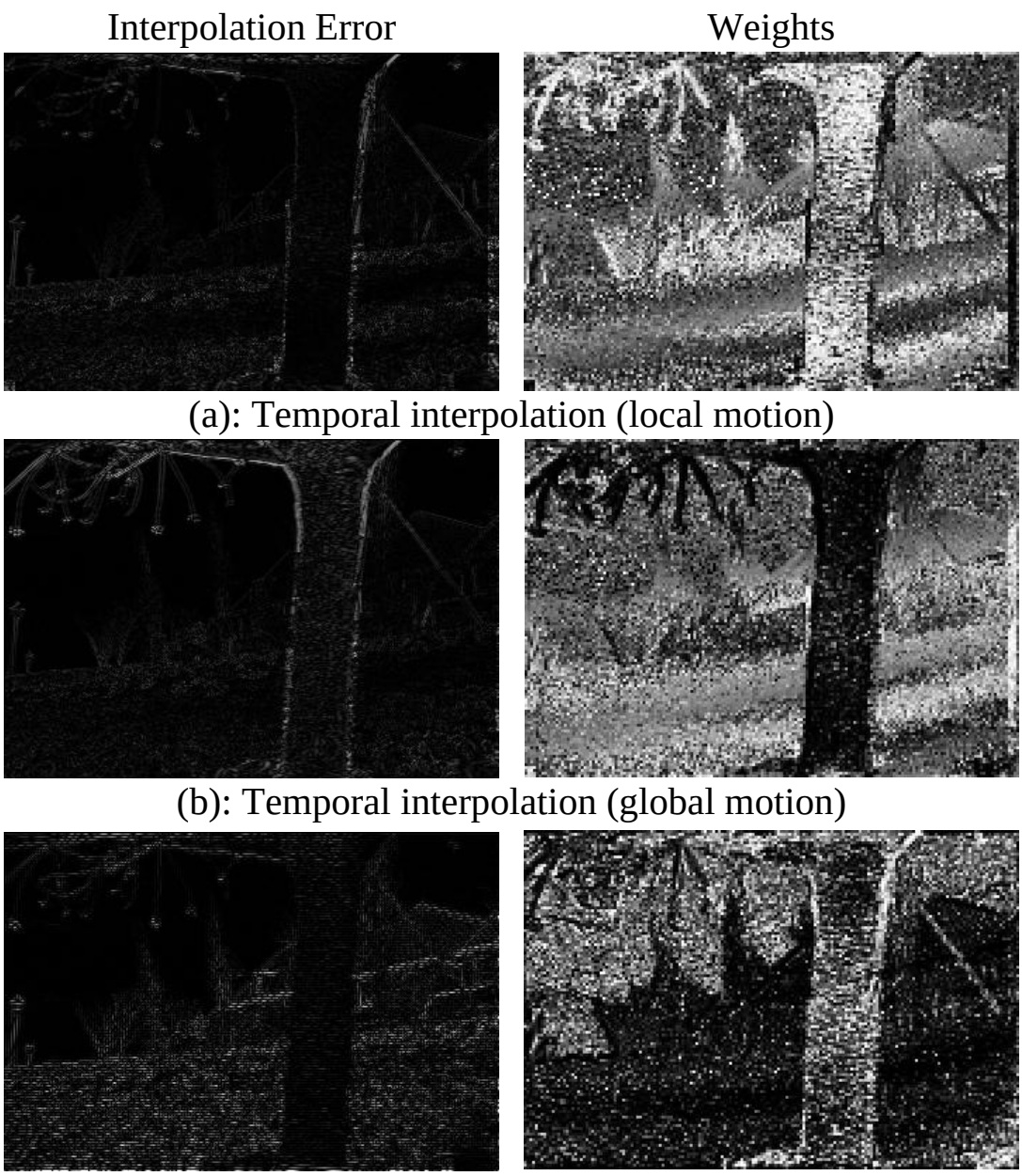


Figure 6.27: [Left]: Interpolation error, [Right]: LMMSE weights. From top to bottom, (a) temporal interpolation using LM, (b) temporal interpolation using GM and (c) spatial interpolation.

6.3.2.1. LMMSE Weights Visualization

Before showing the objective and subjective SR reconstruction results, we first illustrate the LMMSE weights on a sample frame from the *Flower Garden* SIF sequence encoded at 512 kbps. Figure 6.27 shows interpolation errors (left column) and LMMSE weights (right column) for each 4×4 block in the frame. Weight value is proportional to the brightness of a particular block; the brighter the block, the higher the weight.

Note that LM weights w_{TL} tend to be high in the textured foreground areas of the tree trunk where MVs from the bitstream are expected to be fairly accurate. On the other hand, GM weights w_{TG} tend to be high in the background regions where global motion parameters provide accurate interpolation. Finally, spatial weights w_S tend to be high in the smooth areas (e.g. the sky) and boundary regions between foreground and background (e.g. boundary of the tree trunk) where both local and global motion may provide less accurate results.

(a)

(b)

(c)

(d)

Figure 6.28: SR frames produced by the proposed method: (a) *Flower Garden* (frame #3), (b) *Coastguard* (frame #31), (c) *City* (frame #8) and (d) *Mobile Calendar* (frame #40).

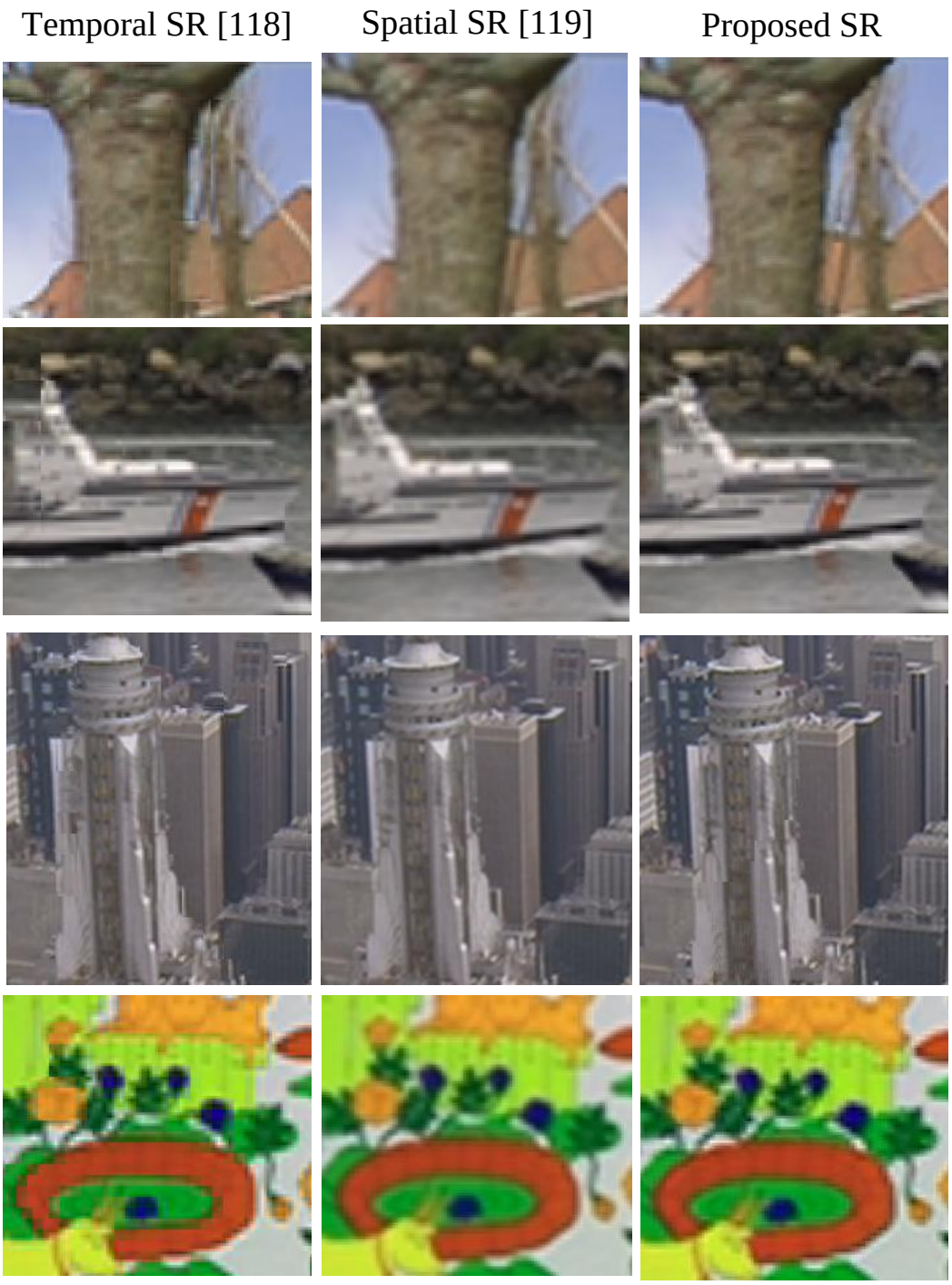


Figure 6.29: SR frames generated by: [Left]: temporal interpolation [118], [Middle]: spatial interpolation [119], and [Right]: proposed method.

6.3.2.2. SR Quality Evaluation

Figure 6.28 shows several SR frames reconstructed by the proposed method from *Flower Garden*, *Coastguard*, *City*, and *Mobile Calendar*. For detailed visual comparison a zoom-in to the regions outlined in red is further shown in Fig. 6.29 for the methods from [118], [119] and the proposed method.

The top row in Fig. 6.29 shows SR frames of *Flower Garden*, reconstructed using the temporal interpolation method from [118] (left), spatial interpolation [119] (middle), and the proposed approach (right). The frame produced by the method from [118] suffers from blocking artefacts along the foreground boundary, particularly in the bottom left portion of the tree trunk. Meanwhile, the method from [119], which relies on spatial interpolation, produces a frame that is free of blocking artefacts, but blurs out the texture inside the tree trunk and rooftops in the background. The proposed method provides a better balance between spatial and temporal interpolation, preserving texture detail simultaneously with suppression of blocking artefacts along foreground boundaries. Similar behaviour can be seen in the other frames in Fig. 6.29.

Finally, for objective evaluation, we compute the luminance Peak Signal-to-Noise Ratio (PSNR) values of SR frames reconstructed by the mentioned three approaches from compressed bitstreams by comparing them to the uncompressed original frames at the SR resolution. Figure 6.30 shows the results for the first 50 frames of the four test sequences. As seen in the figure, the proposed method consistently outperforms the other two methods by a significant margin. The advantage over the method in [118] is 1–3dB on most frames, while the advantage over the method in [119] is 2–6dB. The advantage of both the proposed method and that of [118] over the method in [119] is especially high on *City*, due to the large amount of detail that tends to be blurred out by the method in [119]. The average PSNR results are listed in Table 6.3. In terms of the average PSNR across all sequences, the proposed SR method outperforms the method from [118] by about 1.4dB and that of [119] by about 3dB.

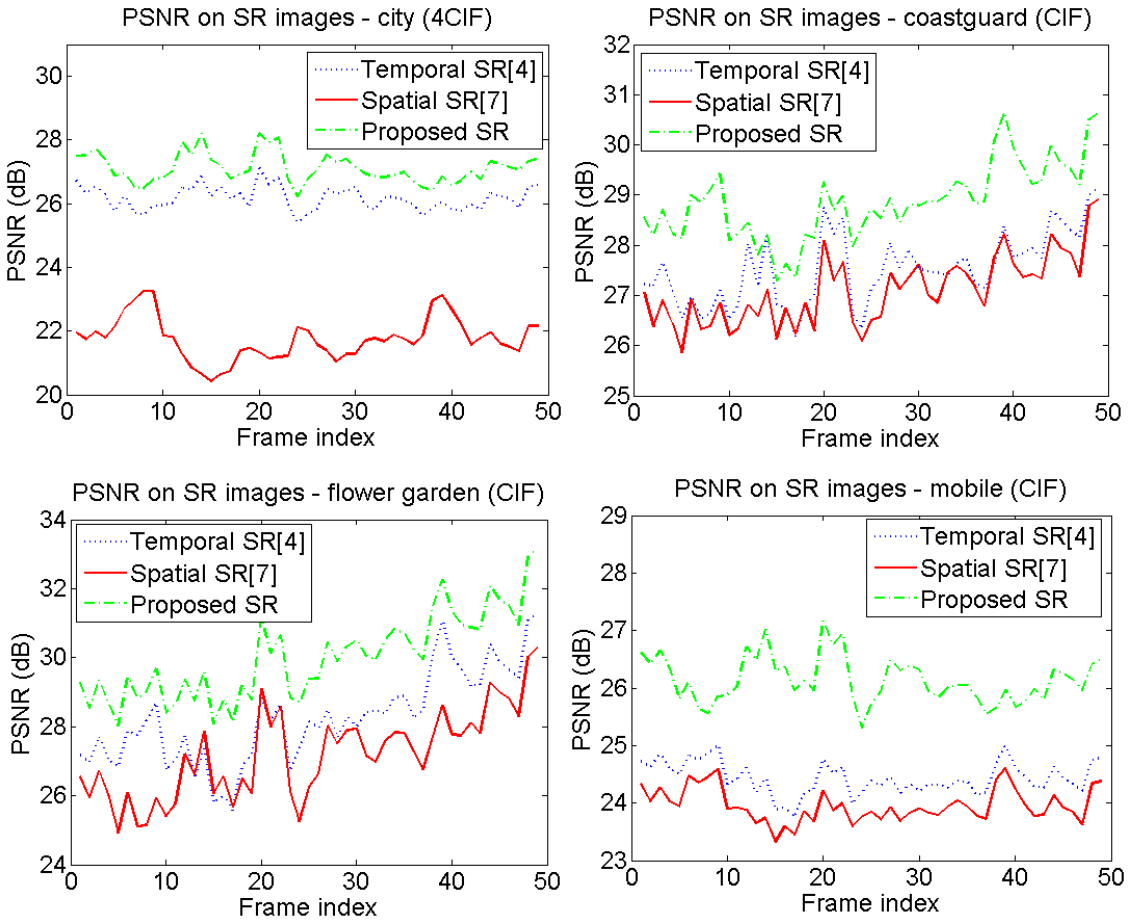


Figure 6.30: PSNR (dB) on the 50 SR frames of *City*, *Coastguard*, *Flower Garden* and *Mobile Calendar*.

Table 6.3: SR performance, average PSNR in dB.

Sequence	Proposed	Temporal SR [118]	Spatial SR [119]
Flower Garden	30.00	28.25	27.19
City	27.11	26.16	21.75
Coastguard	28.86	27.51	27.10
Mobile Calendar	26.14	24.43	23.95
Average	**28.03**	**26.59**	**25.00**

7. Conclusion and Future Directions

In this thesis, we addressed the problem of motion modeling and object segmentation in compressed video, with applications in predicting, interpolating, and resizing video frames in digital communications. Specifically, we assessed, designed and optimized various schemes for global and local motion modeling, and for extracting moving regions from compressed video. We designed a framework to couple global motion estimation and motion segmentation, thereby enabling a system to obtain global motion information and object segmentation simultaneously. Finally, we demonstrated several uses of these techniques in various application scenarios including predictive video decoding, frame rate up-conversion, and super-resolution.

Since the ability of existing MV-based GME schemes to estimate camera motion can be affected by inaccurate MVs, we analyzed the nature of MV outliers and approached outlier removal by categorizing them into Type-1 and Type-2 outliers, caused by noise and foreground objects, respectively. We then proposed a rejection cascade for Type-1 outlier removal and iterative object segmentation for Type-2 outlier removal. The performance of these outlier removal techniques was evaluated by incorporating them into a GME framework, and the improvements compared to state-of-the-art GME approaches have been demonstrated.

To extract object information from standard-compliant compressed video stream, we proposed moving object/region segmentation by making use of the information existing in the bitstream, and avoiding full decoding as much as possible. We looked at the issue of localizing object/region boundaries, and proposed multiple solutions in compressed domain and joint compressed-pixel domain. We evaluated these approaches in terms of computational complexity and segmentation accuracy. Several manually segmented video sequences have also been created in the process and publically released for future research.

To illustrate the application of motion modeling and segmentation in video communications, we dedicated a chapter to elaborate frame synthesis procedures based on motion and object information. We proposed both object-based and pixel-based synthesis approaches, where objected-based frame composition can be achieved by using either boundary matching or object ordinal depth information.

Finally, we looked into various application scenarios that can potentially benefit from techniques developed in motion modeling and objection/region segmentation. Applications are categorized into frame prediction, frame interpolation, and frame resizing. As packet loss and network jitter are two main problems of a practical communication system, we demonstrated that frame prediction can be an attractive solution to reduce video frame rendering delay. In order to save communication bandwidth, video temporal and spatial resolutions have to be traded off. This problem can be mitigated by applying frame temporal interpolation and spatial resizing. In all these scenarios, motion modeling and segmentation play fundamental roles, and reducing their computational expense by directly making use of existing compressed domain information can have a profound impact in real-time system design and deployment.

7.1. Future Work

Conventional motion estimation introduces a number of inaccurate MVs into the encoded MV field. This has a direct impact on decoder-side motion modeling and segmentation performance. Such errors can be even more significant if motion interpolation or extrapolation is required in a particular application. We have exploited various techniques to accommodate MV inaccuracy, however, these methods were performed independently in each frame, and have been considered in the context where the encoder is oblivious to such requirements at the decoder side.

In future work, we plan to extend this research to the encoder side and incorporate object tracking across frames. On the application side, we plan to develop an adaptive video jitter buffer that employs the proposed schemes for real life

performance evaluation. In the rest of this section, we address each of these new contexts, the challenges and research opportunities they bring about.

7.1.1. Object Segmentation Assisted Motion Estimation

To improve application-specific decoder-side segmentation performance, we consider adjusting motion estimation on the encoder side to help achieve this goal. It is a better approach than motion re-estimation at the decoder, since video compression usually loses information permanently during quantization, which is impossible to restore subsequently.

Joint motion estimation and segmentation have been studied for a number of years. A classic reference in this area is [34], however, in this approach the authors did not consider global motion. An alternative approach that takes global motion into account is background subtraction-based moving object segmentation. Again, there is a wealth of existing literature in this area (e.g. video object segmentation robust background modeling) [127] [128]. Once a background mosaic is gradually established, we can use background subtraction technique to segment moving regions. The foreground objects are detected as the difference between the current frame and an image of the scene's background. Motion estimation is then performed within each object, background and foreground, and block partitions are adjusted to follow object boundaries as much as possible. MVs in each moving object will be less likely contaminated by neighbouring objects. Even though object information (i.e. background mosaic) is not transmitted to the decoder, accurate motion information will lead to better decoder-side motion segmentation.

The final issue in this approach is to justify the impact on compression efficiency and encoding complexity. As we stated earlier, such an approach is suitable in a content-based video processing, and the system can be optimized to trade off compression efficiency versus segmentation performance for a specific task.

7.1.2. Object Segmentation and Tracking

Due to the lack of object rigidity and motion field reliability, maintaining consistency of segmentation across multiple frames is a major problem in video object

segmentation. Particularly when segmentation is used for long-term statistical event detection and analysis, over- and under-segmentation poses a serious challenge as it leads to unreliable analytics. We proposed motion segmentation using Markov Random Field (MRF) to mitigate this issue, but the reliability can be compromised by the fact that segmentation is performed independently in each frame.

In this context, a good framework for improving consistency is to combine object segmentation and tracking, where segmentation is used to localize all the objects and detect newly appearing objects, while tracking is responsible for predicting the number of objects and refining objects' moving trajectory.

One challenge of such a combined segmentation and tracking approach is to obtain an initial segmentation map. One simple solution is to construct a supervised system, i.e. let the user decide which objects are relevant and manually segment them at the beginning of a video sequence, however, such solution is impractical if the number of objects of interest is large. We propose a framework that consists of object acquisition and object tracking, where object acquisition is responsible for forming an initial segmentation map that is subsequently used for tracking.

- In the stage of object acquisition, a set of frames is used to coarsely detect objects, which might be over-segmented, and a filtering process is adopted to remove/merge objects to finalize the segmentation map under certain constrains.
- In the stage of object tracking, the main task is to deal with object occlusion, object aging (moving out of the scene), and object detection (moving into the scene). This stage would require sophisticated procedures for computing moving trajectories and maintaining object shape.

The incorporation of object tracking would significantly benefit frame prediction applications. Particularly, object occlusion information will lead to better visual quality of synthesized frames.

7.1.3. Adaptive Video Jitter Buffer Design

The challenges here arise from the need to support a wide variety of video communication systems with various characteristics and capabilities, such as various video coding standards, communication protocols, network conditions, and so on. An important challenge involves designing a video decoding system with adaptive video rendering delay, frame rate, and frame resolution, and developing high-quality frame prediction techniques.

The evaluation of frame prediction in Chapter 6.1 assumed a constant delay reduction, which is not suitable for real life applications as network conditions change constantly. The performance of a video communication system is mainly determined by packet loss and network jitter, thus the key challenge is to maintain high video quality at low end-to-end delay. Predictive decoding can be ideally used here by adopting adaptive prediction depth, and renders high quality video with a much reduced delay. Further application of predictive decoding can be extended to whole frame concealment, which deals with cases that a frame gets lost during communication or arrive late. With combination of techniques developed for frame interpolation and resizing, the future research goal is to achieve better vide communication experience within one-to-one video chat or multi-party video conferencing framework.

7.1.4. Conclusion

In conclusion, in our future work we plan to explore techniques for improving decoder-side segmentation performance, and design practical video rendering systems that operate jointly with the jitter buffer. The main thrust of our research will involve accurate motion modeling and segmentation, motivated by real applications, to offer overall better video quality and video object interaction performance.

We expect to develop schemes for content-based video applications that are cost-effective, taking into account the state of technology, the network characteristics, and the needs of realistic applications. We also expect to develop a system architecture and design a video rendering system that is adaptive to network conditions. The design will be specific to industry needs, such as support for low-bandwidth low-delay communication.

References

[1] C. Stiller and J. Konrad, “Estimating motion in image sequences,” *IEEE Signal Processing Mag*., vol. 16, no. 6, pp. 70-91, Jul. 1999.

[2] F. Dufaux and J. Konrad, “Efficient, robust and fast global motion estimation for video coding,” *IEEE Trans. Image Process*., vol. 9, no. 3, pp. 497–501, Mar. 2000.

[3] Y. Su, M.-T. Sun, and V. Hsu, “Global motion estimation from coarsely sampled motion vector field and the applications,” *IEEE Trans. Circuits Syst. Video Technol*., vol. 15, no. 2, pp. 232-242, Feb. 2005.

[4] MPEG-4 Video Verification M Krutz, M. Frater, M. Kunter, and T. Sikora, “Windowed image registration for robust mosaicing of scenes with large background occlusions,” *Proc. IEEE ICIP’06*, pp. 353-356, Oct. 2006.

[5] Model Version v18.0, ISO/IEC JTC1/SC29/WG11 Coding of Moving Pictures and Audio N3908. Pisa, Italy, Jan. 2001.

[6] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Second Edition. Cambridge University Press, March 2004.

[7] H. Xu, A. A. Younis, and M. R. Kabuka, “Automatic moving object extraction for content-based applications,” *IEEE Trans. Circuits Syst. Video Technol*., vol. 14, no. 6, pp. 796–812, Jun. 2004.

[8] B. Qi, M. Ghazal, and A. Amer, “Robust global motion estimation oriented to video object segmentation,” *IEEE Trans. Image Processing*, vol. 17, no. 6, pp. 958-967, Jun. 2008.

[9] M. Kunter, A. Krutz, M. Mandal, and T. Sikora, “Optimal multiple sprite generation based on physical camera parameter estimation,” *Proc. VCIP’07*, vol. 6508 (2), pp. 0B.1-0B.10 Jan. 2007.

[10] H. Alzoubi and W. D. Pan, “Efficient global motion estimation using fixed and random subsampling patterns,” *Proc. IEEE ICIP’07*, pp. 477-480, Sep. 2007.

[11] Smolic, M. Hoeynck, and J.-R. Ohm, “Low-complexity global motion estimation from P-frame motion vectors for MPEG-7 application,” *Proc. IEEE ICIP’00*, pp. 271-274, Sep. 2000.

[12] R. Wang and T. S. Huang, “Fast camera motion analysis in MPEG domain,” *Proc. IEEE ICIP’06*, pp. 353-356, Oct. 2006.

[13] Y.-P. Tan, D. d. Saur, S. R. Kulkarni, and P. J. Ramadge, “Rapid estimation of camera motion from compressed video with application to video annotation,” *IEEE Trans. Circuits Syst. Video Technol*., vol. 10, pp. 133-146, Jun. 2000.

[14] Y. Deng, and B. S. Manjunath, “Unsupervised segmentation of color-texture regions in images and video,” *IEEE Trans. on Pattern Anal. Mach. Intell*., vol. 23, issue 8, pp. 800-810, Aug. 2001.

[15] Z. Kato, T. C. Pong, and J. C. M. Lee. “Color image segmentation and parameter estimation in a Markovian framework,” *Pattern Recognition Letters*, vol. 22, pp. 309--321, March 2001.

[16] W. Zeng, J. Du, W. Gao, and Q. Huang, “Robust moving object segmentation on H.264/AVC compressed video using the block-based MRF model,” *Real-Time Imaging*, vol. 11, pp. 290-299, Jun. 2005.

[17] J. Wang and E. Adelson, “Representing moving images with layers,” *IEEE Trans. Image Processing*, vol. 3, pp. 625-638, Sept. 1994.

[18] R. V. Babu, K. R. Ramakrishnan, and S. H. Srinivasan, “Video object segmentation: a compressed domain approach,” *IEEE Trans. Circuits Syst. Video Technol*. vol. 14, no. 4 , pp. 462-474, 2004.

[19] N. Vasconcelos and A. Lippman, “Empirical Bayesian motion segmentation,” *IEEE Trans. on Pattern Anal. Mach. Intell*., vol. 2, issue 2, pp. 217-221, Feb. 2001.

[20] Z. Liu, Y. Lu, and Z. Zhang, “Real-time spatiotemporal segmentation of video objects in the H.264 compressed domain,” *J. Visual Communication and Image Representation*, vol. 18, issue 3, pp. 275-290, Jun. 2005.

[21] J. Besag, “On the statistic analysis of dirty pictures,” *J. Royal Statist*. Soc. B. vol. 48, no. 3, pp. 259-302, 1986.

[22] M. Fischler and R. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Comm. ACM*, vol. 24, no.6, pp. 381-395, Jun. 1981

[23] Dante and M. Brookes, “Precise real-time outlier removal from motion vector fields for 3D reconstruction,” *Proc. IEEE ICIP’03*, pp. 393-396, Sep. 2003.

[24] S. Tubaro and S. Rocca, “Motion field estimators and their application to image interpolation,” in *Motion Analysis and Image Sequence Processing* (M. I. Sezan and R. L. Lagendijk, Eds), pp. 153-187, Kluwer, 1993.

[25] D. Farin, *Automatic Video Segmentation Employing Object/Camera Modeling Techniques*, Ph.D. Thesis, Technische Universiteit Eindhoven, Netherlands, 2005.

[26] M. Haller, A. Krutz, and T. Sikora, “Evaluation of pixel- and motion vector-based global motion estimation for camera motion characterization,” *Proc. IEEE WIAMIS’09*, pp. 49-52, May 2009.

[27] Y.-M. Chen, I. V. Bajić, and P. Saeedi, “Coarse-to-fine moving region segmentation in compressed video,” *Proc. IEEE WIAMIS'09*, pp. 45-48, London, UK, May 2009.

[28] Y.-M. Chen and I. V. Bajić, “Motion vector outlier rejection cascade for global motion estimation,” *IEEE Signal Processing Letters*, vol. 17, no. 2, pp. 197-200, Feb. 2010.

[29] Y. Su and M.-T. Sun, “A non-iterative motion vector based global motion estimation algorithm,” *Proc. IEEE ICME'04*, pp. 703-706, Aug. 2004.

[30] D. Wang and L. Wang, “Global motion parameters estimation using a fast and robust algorithm,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, no. 5, pp. 823-826, 1997.

[31] C. Poppe, S. D. Bruyne, T. Paridaens, P. Lambert, R. Van De Walle, “Moving object detection in the H.264/AVC compressed domain for video surveillance applications,**”** *Journal of Visual Communication and Image Representation*, vol. 20, Issue 6, pp. 428-437, Aug. 2009.

[32] H. Hadizadeh and I. V. Bajić, "NAL-SIM: An interactive simulator for H.264/AVC video coding and transmission," *Proc. IEEE CCNC'10*, Las Vegas, NV, Jan. 2010.

[33] L. Cloutier, A. Mitiche, and P. Bouthemy, “Segmentation and estimation of image motion by a robust motion,” *Proc. IEEE ICIP'94*, pp. 805-809, Nov. 1994.

[34] M. M. Chang, A. M. Tekalp, and M. I. Sezan, “Simultaneous motion estimation and segmentation,” *IEEE Trans. Image Processing,* vol. 6, no. 9, pp. 1326-1333, Sep. 1997.

[35] ISO/IEC FDIS 14496-2 Visual, ISO/IEC JTC1/ SC29/ WG11 N2520, Vancouver, Canada, Nov. 1998.

[36] T. Wiegand, G. J. Sullivan, G. Bjontegaard and A. Luthra, “An overview of the H.264/AVC video coding standard,” *IEEE Trans. Circuits Syst. Video Technol.*, pp. 560-576, vol. 13, Jul. 2003.

[37] B. Girod, “Motion-Compensation prediction with fractional-pel accuracy,” *IEEE Trans. Communications*, pp. 604-612, vol. 41, Apr. 1993.

[38] J. Jain and A. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Trans. Commun.*, vol. COMM-29, pp.1799-1808, Dec. 1981.

[39] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, “Motion compensated interframe coding for video conferencing," *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, Nov. 29-Dec. 3 1981, pp. G5.3.1-5.3.5.

[40] R. Srinivasan and K. R. Rao, “Predictive coding based on efficient motion estimation," *IEEE Trans. Commun.*, vol. COMM-33, pp. 888-896, Aug. 1985.

[41] M. Ghanbari, “The cross-search algorithm for motion estimation," *IEEE Trans. Commun.*, vol. 38, pp. 950-953, July 1990.

[42] L. M. Po and W. C. Ma, “A novel four-step search algorithm for fast block motion estimation,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 313-317, June 1996.

[43] L. K. Liu and E. Feig, “A block-based gradient descent search algorithm for block motion estimation in video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 419-423, Aug. 1996.

[44] S. Zhu and K.-K. Ma, “A new diamond search algorithm for fast block matching motion estimation," *IEEE Trans. Image Processing*, vol. 9, pp. 287-290, Feb. 2000.

[45] C. Zhu, X. Lin and L.P. Chau, “Hexagon-Based Search Pattern for Fast Block Motion Estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, pp. 349-355, Aug. 1996.

[46] G. J. Sullivan and R. L. Baker, “Rate-distortion optimized motion compensation for video compression using fixed or variable size blocks," *Proc. GLOBECOM'91*, Phoenix, AZ, pp. 85-90, Dec. 1991.

[47] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini and G. J. Sullivan, “Rate-constrained coder Control and Comparison of Video Coding Standard", *IEEE Trans. Circuit Syst. Video Technol.*, vol. 13, pp.688-703, Jul. 2003.

[48] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” *Proc. IEEE CVPR’01*, pp. 511-518, Dec. 2001

[49] H. Alzoubi and W. D. Pan, “Efficient global motion estimation using fixed and random subsampling patterns,” *Proc. IEEE ICIP’07*, pp. 477-480, Sep. 2007

[50] N. Kawamura, M. Yoshimura, and S. Abe, “Image query by multiresolution spectral histograms,” *Proc. IEEE CIMCA-IAWTIC’05*, vol. 2, pp. 660-665, Nov. 2005.

[51] D. Zhong and S. F. Chang, “An integrated approach for content-based video object segmentation and retrieval,” *IEEE Trans. Circuits Syst. Video Technol.,* vol. 9, pp. 1259-1268, Dec. 1999.

[52] Y. M. Chen and I. V. Bajić, “Predictive decoding for delay reduction in video communications,” *Proc. IEEE Globecomm’07*, pp. 2053-2057, Washington, DC, Nov. 2007.

[53] G. Iannizzotto, and L. Vita, “Fast and accurate edge-based segmentation with no contour smoothing in 2-D real images,” *IEEE Trans. Image Processing,* vol. 9, issue, 7, pp. 1232-1237, Jul. 2000.

[54] Y. Li, D. Lu, X. Lu, and J. Liu, “Interactive color image segmentation by region growing combined with image enhancement based on Bezier model,” *Proc. IEEE Intl. Conf. Image and Graphics,* pp. 96-99, Dec. 2004.

[55] H. Y. Chung, Y. L. Chin, K. Wong, K. P. Chow, T. Luo, and S. K. Fung, “Efficient block based motion segmentation method using motion vector consistency,” *Proc. IAPR Conf. on Machine Vision Applications*, pp. 550 – 553, May 2005.

[56] D. Cremers, and S. Soatto, “Variational space-time motion segmentation,” *Proc. IEEE ICCV,* vol. 2, pp. 886-893, Oct. 2003.

[57] R. Ahmed, G. C. Karmakar and L. S. Dooley, “Incorporation of texture information for joint spatio-temporal probabilistic video object segmentation,” *Proc. IEEE ICIP*, vol. 6, pp. 293-296, Sep. 2007.

[58] A. Briassouli, V. Mezaris, and I. Kompatsiaris, “Joint motion and color statistical video processing for motion segmentation,” *Proc. IEEE ICME*, pp. 2014-2017, Jul. 2007.

[59] X. Song and G. Fan, “Joint key-frame extraction and object segmentation for content-based video analysis,” *IEEE Trans. Circuits Syst. Video Technol.,* vol. 16, no. 7, pp.904-914, Jul. 2006.

[60] Y.M. Chen, *Predictive decoding for delay reduction in video communications*, M.A.Sc. Thesis, Simon Fraser University, Burnaby, BC, Dec. 2007. [Online] Available: http://troy.lib.sfu.ca/record=b5372807~S1a

[61] J. Y. A. Wang and E. H. Adelson, “Representing moving images with layers,” *IEEE Trans. Image Processing*, vol. 3, no. 5, p.625-638, Sep. 1994.

[62] G. D. Borshukov, G. Bozdagi, Y. Altunbasak, and A. M. Tekalp, “Motion segmentation by multistage affine classification,” *IEEE Trans. Image Processing*, vol. 6, no. 11, pp. 1591-1594, Nov. 1997.

[63] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, “An efficient k-means clustering algorithm: Analysis and implementation,” *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 24, no. 7, pp. 881-892, Jul. 2002.

[64] H. Y. Chung, Y. L. Chin, K. Wong, K. P. Chow, T. Luo, and S. K. Fung, “Efficient block based motion segmentation method using motion vector consistency,” *Proc. IAPR Conference on Machine Vision Applications*, pp. 550 – 553, May 2005.

[65] Z. Liu, Z. Zhang, and L. Sheng, “Moving object segmentation in the H.264 compressed domain,” *Optical Engineering*, vol. 46, no. 1, pp. 017003: 1-5.

[66] Brouard, F. Delannay, V. Ricordel, and D. Barba, “Spatio-temporal segmentation and regions tracking of high definition video sequences based on a Markov Random Field model,” *Proc. IEEE ICIP’08*, pp. 1552-1555, Dec. 2008.

[67] W. Zeng, J. Du, W. Gao, and Q. Huang, “Robust moving object segmentation on H.264/AVC compressed video using the block-based MRF model,” *Real-Time Imaging*, vol. 11, pp. 290-299, Jun. 2005.

[68] M. Ritch and N. Canagarajah, “Motion-based video object tracking in the compressed domain,” *Proc. IEEE ICIP’07,* vol. 6, pp. VI-301-VI-304, Oct. 2007.

[69] R. V. Babu, K. R. Ramakrishnan, and S. H. Srinivasan, “Video object segmentation: a compressed domain approach,” *IEEE Trans. Ciruits Syst. Video Technol*. vol. 14, no. 4 , pp. 462-474, Apr. 2004.

[70] N. Vasconcelos and A. Lippman, “Empirical Bayesian motion segmentation,” *IEEE Trans. on Pattern Anal. Mach. Intell*., vol. 2, issue 2, pp. 217-221, Feb. 2001.

[71] X. Shi, Z. Zhang, L. Shen, “Multiresolution segmentation of video objects in the compression domain,” *Optical Engineering.,* vol. 46, no. 9, 097401, Sep. 2007.

[72] R. Ewerth, M. Schwalb, P. Tessmann, and B. Freisleben, “Segmenting moving objects in MPEG videos in the presents of camera motion,” *Proc. ICIAP’07*, pp. 92-96, Sep. 2007.

[73] Y.-M. Chen, I. V. Bajić, and P. Saeedi, "Motion segmentation in compressed video using Markov random fields," *Proc. IEEE ICME'10*, pp. 760-765, Singapore, Jul. 2010.

[74] Z. Kato, “Segmentation of color images via reversible jump MCMC sampling,” *Image and Vision Computing*, vol. 26, issue 3, pp. 361-371, Mar. 2008.

[75] Y.-M. Chen, I. V. Bajić, and P. Saeedi, "Moving region segmentation from compressed video using global motion estimation and Markov random fields," *IEEE Trans. Multimedia*, vol. 13, no. 3, pp. 421-431, Jun. 2011. (Special Issue on ICME 2010).

[76] *Overview of the MPEG-4 Standard*, V.18, ISO/IEC JTC11/SC29/WG11 N4030, Mar. 2001.

[77] S.-F. Chang, T. Sikora, and A. Puri, “Overview of the MPEG-7 standard,” *IEEE Trans. Circuits Syst. Video Technol.*, vol.11, no. 6, pp. 688-695, Jun. 2001.

[78] V. Mezaris, I. Kompatsiaris, N. V. Boulgouris, and M. G. Strintzis, “Real-time compressed-domain spatio-temporal segmentation and ontologies for video indexing and retrieval,” *IEEE Trans. Circuits Syst. Video Technol.*, vol.14, no. 5, pp. 606-621, May 2004.

[79] M. Said Allili and D. Ziou, “Active contours for video object tracking using region, boundary and shape information,” *Journal of Signal, Image and Video Processing*, vol. 1, no. 2, pp. 101-117, Jun. 2007.

[80] R. Achanta, M. Kankanhalli, and P. Mulhem, “Compressed domain object tracking for automatic indexing of objects in MPEG home video,” *Proc. IEEE ICME’02*, vol. 2, pp. 61-64, Nov. 2002.

[81] Sukmarg and K. R. Rao, “Fast object detection and segmentation in MPEG compressed domain,” *Proc. TENCON’00*, vol. 3, pp. 364-368, Sep. 2000.

[82] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, Third Edition, Prentice Hall, 2008.

[83] S. Ogale, C. Fermuller, and Y. Aloimonos, “Motion segmentation using occlusions,” IEEE Trans. on *Pattern Anal. Mach. Intell*., vol. 27, no. 6, Jun. 2005.

[84] Y.-L Chang, C.-Y. Fang, L.-F Ding, S.-Y. Chen, and L.-G. Chen, “depth map generation for 2D-to-3D conversion by short-term motion assisted color segmentation,” *Proc. IEEE ICME’07*, pp. 1958-1961, 2007.

[85] Y. Feng, J. Jayaseelan, and J. Jiang, “Cue based disparity estimation for possible 2D to 3D video conversion,” *Proc. VIE’06*, pp. 384-388, Sept. 2006.

[86] D. Zhong and S.-F. Chang, “Long-term moving object segmentation and tracking using spatio-temporal consistency,” *Proc. IEEE ICIP’01*, Greece, vol. 2, pp. 57-60, Oct. 2001.

[87] S. Belfiore, M. Grangetto, E. Magli, and G. Olmo, “An error concealment algorithm for streaming video,” *Proc. IEEE ICIP’03*, vol. 3, pp. 649-652, Sep. 2003.

[88] P. Baccichet, D. Bagni, and A. Chimienti, “Frame concealment for H.264/AVC decoders,” *IEEE Trans. Consumer Electronics*, vol. 51, no. 1, pp. 227-233, Feb. 2005.

[89] X. Li, “Least-square prediction for backward adaptive video coding,” *EURASIP Journal on Applied Signal Processing,* vol. 2006, pp. 1-13, Feb. 2006.

[90] J. E. Boyd, J. Meloche, and Y. Vardi, “statistical tracking in video traffic surveillance,” *Proc. IEEE ICCV’99*, vol. 1, pp. 163-168, 1999.

[91] Y. Zhang, Z. Zhai, X. Nie, C. Ma and F. Zuo, “an extended self-adaptive Kalman filtering object motion prediction model,” *Proc. IIH-MSP’08*, pp. 421-424, Aug. 2008.

[92] Y.-M. Chen and I. V. Bajić, "Predictive video decoding based on ordinal depth of moving regions," *Proc. IEEE ICC'10*, Cape Town, South Africa, May 2010.

[93] Y.-M. Chen and I. V. Bajić, "Region-based predictive decoding of video," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 3, pp. 452-457, Mar. 2010.

[94] R. C. Kordasiewicz, M. D. Gallant, and S. Shirani, “Affine motion prediction based on translational motion vector,” *IEEE Trans. Circuits Syst. Video Technol.,* vol. 17, no. 10, pp. 1388-1394, Oct. 2007.

[95] B. D. Choi, J. W. Han, C. S. Kim, and S. J. Ko, “Motion-compensated frame interpolation using bilateral motion estimation and adaptive overlapped block motion compensation,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 4, pp. 407-416, Apr. 2007.

[96] M. Silveira and M. Piedade, “Variable block sized motion segmentation for video coding,” *Proc. IEEE ISCAS’97,* vol. 2, pp. 1293-1296, Jun. 1997.

[97] Katsaggelos and N. Galatsanos, *Signal Recovery Techniques for Image and Video Compression and Transmission*, Kluwer Academic Publishers, 1998.

[98] M. J. Chen, L. G. Chen, and R. M. Weng, “Error concealment of lost motion vectors with overlapped motion compensation,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, no. 3, pp. 560-563, Jun. 1997.

[99] Y. Chen, O. C. Au, C.-W. Ho, and J. Zhou, “Spatio-temporal boundary matching algorithm for temporal error concealment,” *Proc. IEEE ISCAS’06*, pp. 686-689, May 2006.

[100] Y.-M. Chen and I. V. Bajić, “Predictive video decoding using GME and motion reliability,” *Proc. SPIE Applications of Digital Image Processing XXXIV*, Vol. 8135, San Diego, CA, Aug. 2011.

[101] Y.-M. Chen and I. V. Bajić, "Spatio-temporal super-resolution from compressed video employing global and local motion," *Proc. IEEE PacRim'11*, pp. 907-912, Victoria, BC, Aug. 2011.

[102] S. Belfiore, M. Grangetto, E. Magli, and G. Olmo, “Concealment of whole-frame losses for wireless low bit-rate video based on multiframe optical flow estimation,” *IEEE Trans. Multimedia*, vol. 7, no. 2, pp. 316-329, Apr. 2005.

[103] P. Baccichet, D. Bagni, and A. Chimienti, “Frame concealment for H.264/AVC decoders,” *IEEE Trans. Consumer Electronics*, vol. 51, no. 1, pp. 227-233, Feb. 2005.

[104] P. Baccichet and A. Chimienti, “A low complexity concealment algorithm for the whole-frame loss in H.264/AVC,” *Proc. IEEE MMSP’04*, pp. 279-282, Oct. 2004.

[105] E. Quacchio, E. Magli, G. Olmo, P. Baccichet, and A. Chimienti, “Enhancing whole-frame error concealment with an intra motion vector estimator in H.264/AVC,” *Proc. IEEE ICASSP’05*, vol. 2, pp. 329-332, Mar. 2005.

[106] S. C. Sui, F. Liu and P. Xue, “Whole-frame error concealment with improved backward motion estimation for H.264 decoders,” *IEEE Fourth International Conference on Image and Graphics*. pp. 279-284, Aug. 2007.

[107] M. Kalman, E. Steinbach, and B. Girod, “Adaptive media playout for low-delay video streaming over error-prone channels,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 14, no. 6, pp. 841-851, June. 2004.

[108] X. Li, “Least-square prediction for backward adaptive video coding,” *EURASIP Journal on Applied Signal Processing,* vol. 2006, pp. 1-13, Feb. 2006.

[109] C.-H. Kwak and I. V. Bajić, "Error concealment strategies for motion capture data streaming," presented at *IEEE StreamComm*, in conjunction with *IEEE ICME'11*, Barcelona, Spain, Jul. 2011.

[110] Y.-T. Yang, Y.-S. Tung, and J.-L. Wu, “Quality enhancement of frame rate up-conversion video by adaptive frame skip and reliable motion extraction,” *IEEE Trans. Circuits Syst. Video Technol.* vol. 17, no. 12, pp. 1700-1713, Dec. 2007.

[111] A.-M. Huang and T. Q. Nguyen, “A multistage motion vector processing method for motion-compensated frame interpolation,” *IEEE Trans. Image Processing.* vol. 17, no. 5., pp. 694-708, May. 2008.

[112] S.-C. Tai, Y.-R. Chen, Z.-B. Huang, and C.-C. Wang, “A multi-pass true motion estimation scheme with motion vector propagation for frame rate up-conversion applications,” *Journal of Display Technol.* vol. 4, no. 2, pp. 188-197, Jun. 2008.

[113] J. Wang, N. Patel, W. Grosky, and F. Fotouhi, “Video frame rate up conversion under inconsistent camera motion,” *Journal of Multimedia Tools Appl.* vol. 39. no. 3, pp. 329-351, Sept. 2008.

[114] J. Benois-Pinneeau and H. Nicolas, “A new method for region-based depth ordering in a video sequence: application to frame interpolation,” *J. Visual Communication and Image Representation*, vol. 13, issue 3, pp. 363-385, Sept. 2002.

[115] S. C. Park, M. K. Park, and M. G. Kang, “Super-resolution image reconstruction: A technical overview”, *IEEE Signal Processing Magazine*, Vol. 20.3, pp. 21-36, May 2003.

[116] R. R. Schultz and R.L. Stevenson, “Extraction of high resolution frames from video sequences”, *IEEE Trans. Image Processing*, vol. 5, no. 6, pp. 996-1011, June 1996.

[117] S. Peleg, D. Keren and L. Schweitzer, “Improving image resolution using subpixel motion,” *Pattern Recognition Letters*, vol. 5, pp. 223-226, Mar. 1987.

[118] G. M. Callico, A. Nu.nez, R. P. Llopis, R. Sethuraman, and M. O. de Beeck, “A low-cost implementation of super-resolution based on a video encoder”, *Proc. IEEE Annual Conf. of the Industrial Elec. Society*, vol. 2, pp. 1439-1444, Nov. 2002.

[119] J. Zhang, X. Fang, C. Huang, and Y. Tian, “Multispectral image superresolution reconstruction based on B-spline interpolation,” *Proc. SPIE*, Vol. 7494, pp. 1117 – 1121, Oct. 2009.

[120] ITU-T (1988). "H.261 : Video codec for audiovisual services at p x 384 kbit/s - Recommendation H.261 (11/88)," 1988.

[121] ITU-T (1995). "H.263 : Video coding for low bit rate communication,” 1995.

[122] ISO/IEC JTC 1/SC 29 (2009-10-30). "Programme of Work — Allocated to SC 29/WG 11, MPEG-1,” Nov, 2009.

[123] ISO (1996). "ISO/IEC 13818-1:1996 - Information technology -- Generic coding of moving pictures and associated audio information: Systems," 1996.

[124] Y.-M. Chen and I. V. Bajić, "Compressed-domain moving region segmentation with pixel precision using motion integration," *Proc. IEEE PacRim'09*, pp. 442-447, Victoria, BC, August 2009.

[125] J. Youn, M.-T. Sun, and C.-W. Lin, “Motion vector refinement for high performance transcoding,” *IEEE Trans. Multimedia*, vol. 1, no. 1, pp. 30–40, Mar. 1999.

[126] Y.-M. Chen and I. V. Bajić, "A joint approach to global motion estimation and motion segmentation from a coarsely sampled motion vector field," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 21, no. 9, pp. 1316-1328, Sep. 2011.

[127] A. Colombari, A. Fusiello, and V. Murino, “Video object segmentation by robust background modeling,” *Proc. ICIAP'07*, pp. 155-164, sept. 2007.

[128] A. Colombari, A. Fusiello, and V. Murino, “Segmentation and tracking of multiple video objects”. *Pattern Recognition*, vol. 40, pp.1307-1317, April 2007.