INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

NOTE TO USERS

This reproduction is the best copy available.

UMI®

DISSERTATION

IMAGE SEQUENCE SEGMENTATION USING MULTIPLE FEATURES AND EDGE FUSION: ITS ALGORITHM AND VLSI ARCHITECTURE

Submitted by

Jinsang Kim

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

for the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

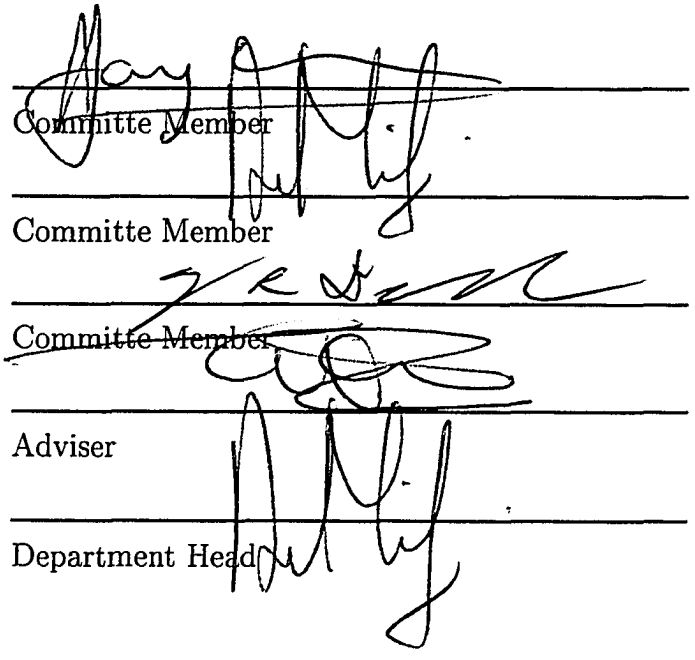
Fall, 2000

COLORADO STATE UNIVERSITY

July 18, 2000

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY JINSANG KIM ENTITLED IMAGE SEQUENCE SEGMENTATION USING MULTIPLE FEATURES AND EDGE FUSION: ITS ALGORITHM AND VLSI ARCHITECTURE BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work



Committe Member

Committe Member

Committe Member

Adviser

Department Head

ABSTRACT OF DISSERTATION

IMAGE SEQUENCE SEGMENTATION USING MULTIPLE FEATURES AND EDGE FUSION: ITS ALGORITHM AND VLSI ARCHITECTURE

Semantic object representation is an important step for digital multimedia applications such as object-based coding, content-based access, and manipulations. This dissertation presents an image sequence segmentation algorithm and its VLSI architecture which provides initial region information for the video coding and the semantic object representation in image sequences. Our objective is to develop a hardware-friendly segmentation algorithm and its architecture by combining static and dynamic features simultaneously in one scheme.

In the initial stage of the algorithm, a multiple feature space is transformed to a label space by using the self-organizing feature maps (SOFM) neural networks. The next stage is an edge fusion in which edge information is incorporated into the neural network outputs to generate more precisely located boundaries of segmentation.

Pixel-based feature vectors consisting of three color, motion, and two texture features are extracted from two frames of an image sequence. These feature vectors are smoothed and normalized. A soft weighting scheme is applied to the normalized features. The weighting scheme suppresses unreliable feature components in a feature vector by making their values low. In order to generate the segmentation label space, the weighted multiple feature space is transformed to the one-dimensional label space using the SOFM neural networks. The oversegmented segmentation labels are further

processed by incorporating edge information in order to generate segmented region boundaries closer to edges. The edge fusion is an iterative region merging process using a similarity criterion consisting of color difference, region geometry, and edge information between two regions. Experimental results for a variety of MPEG image sequences are evaluated and compared with an existing segmentation method to clarify the advantages of the proposed algorithm objectively.

The proposed algorithm differs from existing methods as followings: (1) it can segment textured images with low-dimensional texture features, (2) it leads to more meaningful segmentation region boundaries, and (3) it is easier to be mapped into hardware than existing methods.

Also, this dissertation proposes a VLSI segmentation architecture of the proposed algorithm. The proposed segmentation scheme is mapped into a dedicated hardware system. The dedicated special-purpose system consists of motion estimation, edge detection, edge linking, median and *min* filters, feature normalization and weighting, the systolic feature labeling, and edge fusion subsystems which can be easily mapped into systolic and pipelined architectures. Computational and hardware complexities of the proposed system architecture are estimated in terms of the number of clock cycles, arithmetic components and memory requirement. The proposed VLSI architecture makes it possible to perform image sequence segmentation in real-time.

Jinsang Kim
Department of Electrical and Computer Engineering
Colorado State University
Fort Collins, Colorado 80523
Fall, 2000

ACKNOWLEDGEMENTS

I would like to express my heartfelt thanks to my advisor, Dr. Thomas Wei Chen for his great enthusiasm, constructive criticism, and encouragement throughout my Ph.D. program. His invaluable assistance on work in Hewlett Packard, writing and presenting research results is also greatly appreciated.

I want to thank Dr. Delek L. Lile, Dr. Anura P. Jayasumana, and Dr. J. Ross Beveridge for their invaluable advice, assistance, and encouragement as committee members. I am thankful to Dr. Mahmood Azimi-Sagjadi as temporary committee member. Also, I would like to give special thanks to Dr. Bruce A. Draper in the department of computer science for his suggestion on objective evaluation.

I would like to thank my colleagues and friends, Amjad Hajjar, Geunrae Cho, Alkan Gengiz, Seungik Lee, John Swanson, Dr. David Chen, and Dr. Vonkyong Kim in the VLSI System Architecture Laboratory for their support and suggestion. Also, I thank Korean friends in our department, Dr. Younghoon Kim, Dr. Sanghoon Seo, Yoonggoog Cho, and Sanghoon Lim for their friendship and encouragement.

I am very thankful to President Kyocheol Lee, Vice president Dr. Sanghoon Lee in Korea Telecom, and President Dr. Yongkyung Lee in Korea Telecom Freetel. They had given me financial support through my Ph.D program. Also, I would like thank co-workers in Korea Telecom Research Center for their encouragement. This research could not have been accomplished without their help. I especially appreciate Professor Wonkyung Cho at Kyunghee University. His academic assistance and great encouragement in my college year made me continue to study even when I felt low. I am also thankful to friends of mine who have grown up and have shared friendship with me.

My warmest thanks go to great persons, Mr. Dennis Houska and Mrs. Noreen Houska, for hosting our family during our stay in Fort Collins. They gave us a chance to see, learn, and feel a lot on USA.

I am always thankful for the love and support of my parent-in-law, my brothers, Jinhang and Jinyang, and my sisters, Soowol and Soonim. Thanks, father and mother. I miss you so much, Mom. She had sacrificed her whole life for me, and could not see me from a closer distance during my Ph.D program. All I can remember is their love and patience. Finally, my special thanks go to my beautiful two daughters, Yoonhyung and Boryung, and my lovely wife, Jungeun Lee for their support, trust, and love.

DEDICATION

To the memory of my mother, Whasoon Kim (1928-1999),

to my father, my parent-in-law,

and most especially, my wife, Jungeun Lee.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Background

The rapid development of VLSI and digital signal processing technology has created new digital services and applications in the computer, telecommunication, internet, and broadcasting field whose boundaries become blurred. Applications for digital libraries or content-based storage and retrieval should allow access to video data based on object representations described by attributes such as color, shape, and motion. Mobile multimedia services require high compression performance and object-based coding functionalities which allow for interaction with audio-visual objects within limited bandwidth by allocating a limited bit rate to different semantic parts of a scene and to fit individual needs. Real-time communications like video telephony should have the ability to successfully operate over a wide variety of media including low- and high-mobility wireless systems, LAN, PSTN, and ISDN transmission channels such that the overall end-to-end delay will be relatively small and fairly constant.

Research has been done to define and explore a new paradigm of creating and using digital video [4, 5, 6, 7, 8], and with it comes the emerging digital video applications including virtual reality, enhanced viewing experiences, and creative information services. Some of the digital video applications are enabled by compression standards

like MPEG-2, but most of the compression standards go beyond video compression and simple playback mechanisms. The MPEG-4 standard has been developed for those applications. The MPEG-4 standard specifies a multimedia bit stream syntax and a set of tools and interfaces for designers and builders for a wide variety of multimedia applications [9]. Since MPEG-4 defines a video decoder, the problem of image analysis was avoided by using presegmented video sequences [9, 10]. This decision allows different implementations of object descriptions and object-based video coders. All these applications have one common requirement: *video contents should be represented based on image analysis techniques.*

An image sequence can be hierarchically decomposed into scenes and frames as vision system can represent the layered description of the contents of an image [11]. A scene is a collection of frames in time which are temporally correlated. Therefore, semantically meaningful objects, which a user may want to manipulate, can be extracted using the analysis results of one or more frames in a scene. It is virtually impossible to find semantically meaningful objects in the scene automatically, since a single homogeneous criterion like color or motion cannot extract satisfactory semantic objects, and the definition is vague and domain-dependent. In general, a semantic object contains different color, texture, and motion. For semantic object representation, regions (a homogeneous area segmented by static features) of homogeneous static feature and motion features can be grouped hierarchically as shown in Figure 1.1.

Static features such as gray-level, color, and texture are relatively stable for each region over time. Thus homogeneous static features such as gray value, color, or texture are good homogeneous criterion for the bottom level region segmentation of frames. Moving objects in a sequence, referred to as *intermediate level objects*, consist of one or more bottom level regions, semantically meaningful objects, referred to as

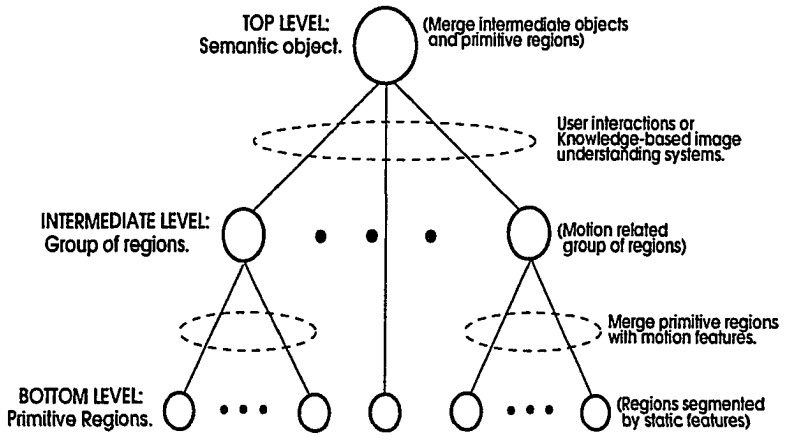
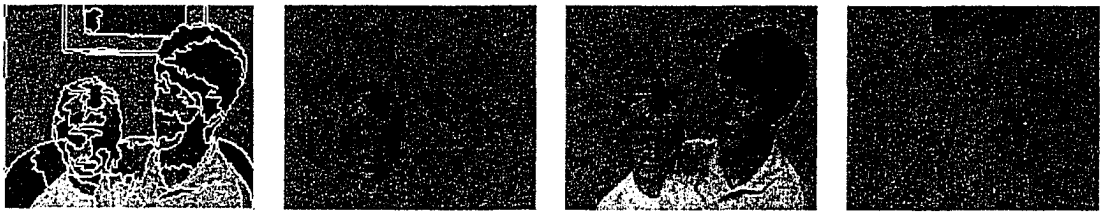


Figure 1.1: The hierarchical video representation model.



(a) (b) (c) (d)

Figure 1.2: The different object representations of *mother&daughter* sequence from a segmentation of a frame. (a) bottom level segmentation, (b) moving object, (c) semantic object, (d) a background object.

top level objects, can be created by grouping intermediate objects or regions of interest in the bottom level regions by human interactions or knowledge-based systems.

Figure 1.2 shows examples that a bottom level region segmentation creates different representations of objects in an image sequence. Figure 1.2 (a) is an example of bottom level region segmentation of a *mother&daughter* frame. By grouping homogeneous motion regions (assume only the daughter is moving for several frames), a moving object, which is useful for coding applications, can be extracted as shown in Figure 1.2 (b). By human interactions, for example, locating the merged region boundaries along the mother and daughter, the foreground of the scene can be ex-

tracted as shown in Figure 1.2 (c) so that the semantic object can be manipulated. Also, by choosing an object of interest like Figure 1.2 (d) and replacing it with another one, we can generate a different background scene. Therefore, *segmentation* is a very important step for representing visual objects.

1.2 General Description of the Problem

Hundreds of segmentation methods have been presented, but there is no unified method which can be considered good for all the images. Most algorithms are specific to the image formation model. Algorithms developed for one class of images may not always be applied to the other classes. The segmentation is still a very challenging topic in the computer vision and image processing community: (1) homogeneity criteria are scene-dependent, (2) no ground-truth segmentation information is available, (3) the number of final segmented regions is usually unknown, and (4) segmentation is the ill-posed problems [12].

For extracting objects in a scene, several techniques [13, 14, 15, 16, 17] rely on only motion features, which is essential in image sequence. However, in general, the results have blurred segmentation boundaries because of the ill-posed problem of motion estimate [4]. Two-phase segmentation approaches [8, 18, 19] for representing moving objects have been proposed by supporting static segmentation. During the initial phase, spatial gray-level or color segmentation of a frame is first performed. In order to extract moving objects, the segmented regions are merged and tracked using an affine motion projection or area operations of dense motion projection of the spatially segmented regions in the second phase. These techniques are more robust than those using only the motion feature, since the boundaries of moving objects coincide with those of static segmentation. However, these techniques ignore the presence of textured regions. Therefore, they produce oversegmentation results when

an image sequence has textured regions. Also, the iterative process and need for priori knowledge of a scene (the number of objects) [8, 7] make the implementation of a real-time segmentation system difficult. The more detail descriptions of the problems in existing algorithms are presented in Sections 4.2 and 5.1. The limitations of the existing methods are summarized as follows:

- The use of unreliable motion during the second phase for grouping regions by an affine motion model may generate erroneously merged regions.
- In general, the presence of textured features in the scene is ignored, and the homogeniety criterion for the spatial segmentation is gray-level or color. Therefore, the results create oversegmentation in the textured area, and their algorithms need a sophisticated processing for more meaningful segmentation results.
- The algorithms lack generality, since they limit the scenes to which they are successfully applied, and the number of test sequences are usually one or two.
- Most of the algorithms do not use edge information. For more meaningful segmentation, edge information needs to be incorporated during segmentation.
- Most of the segmentation algorithms are of a sequential and iterative nature, keeping them from being implemented in real-time systems.
- Most of the algorithms lack in objective evaluations of their segmentation results. It is desirable to provide evaluation information of a segmentation algorithm so that a user for later goal-oriented process can select a segmentation algorithm based on its evaluation data.
- No special-purpose hardware system for image sequence segmentation is presented in the literature. For real-time applications such as video telephony and live broadcasting, segmentation hardware needs to be developed.

1.3 Research Objective and Approaches

The following features are considered to be important in an image sequence segmentation system:

a. Flexibility: The segmentation should deal with static and motion feature in order to cover all kinds of image activity.

b. Reliability: The segmentation should not use unreliable motion features which may lead to segmented region boundaries far from an original image.

c. Accuracy: The algorithm should generate precisely located boundaries of segmented regions.

d. Complexity: The segmentation algorithm should avoid iterative procedures as much as possible without degrading segmentation quality, and as much as possible it should have parallel processing subtasks.

e. Hardware mappability: The algorithm should have hardware-friendly subfunctions which can provide parallel operations with simple arithmetic hardware.

Feature (a) can be implemented by clustering pixel-based multiple features including luminance/chrominance, motion, and texture features. In order to implement feature (b), a criterion is needed to decide whether the estimated motion is reliable or not. This can be done through a controlled feature weighting scheme. Edge information can be incorporated into the segmentation process to implement feature (c). Neural networks can provide fast operations, since trained network weights can be reusable for other feature vectors for image segmentation. In addition, they are parallel in nature with simple arithmetic operations. Therefore, to implement features (d) and (e), neural networks can be used for the segmentation.

The objective of this research is to develop an image sequence segmentation algorithm and its VLSI architecture which provides fast and efficient descriptions of primitive regions or region groups for object-oriented applications of image sequences. First, we propose a segmentation algorithm which can handle the existing limitations as mentioned in Section 1.2 as much as possible. Also, the proposed algorithm is mapped into a VLSI architecture which makes an image sequence segmentation real-time.

The proposed segmentation scheme can be summarized as follows. The selected multiple features consist of static and dynamic features such as three color, motion, and two texture features. A soft weighting scheme suppresses unreliable feature components. A multiple feature space is transformed to one dimensional label space by using the self-organizing feature maps (SOFM) neural networks. Edge information is incorporated into the neural network outputs to generate more precisely located boundaries of segmentation. The proposed algorithm differs from existing methods as following: (1) it can segment textured images with low-dimensional features, (2) it leads to more meaningful segmentation region boundaries, and (3) it is easier to be mapped into hardware than existing methods. The segmentation results are also evaluated and compared with an existing segmentation method using evaluation metrics to clarify the advantages of the proposed algorithm objectively.

In this dissertation, the proposed segmentation algorithm is also mapped into a VLSI architecture for real-time image sequence segmentation. The feature extraction, filtering, normalization, weighting, and labeling subtasks are highly regular tasks and are mapped into systolic and pipelined architectures. The edge fusion subtask involves an iterative region merging process which needs to access an arbitrarily shaped region by index addressing of the region label. The performance of the proposed VLSI segmentation architecture is estimated in terms of the number of clock cycles and hardware components.

1.4 Main Contributions

In general, image and video processing associate with high computation and I/O bandwidths and large storage. Image and video segmentation schemes involve complex subtasks performing different computational characteristics such as feature extraction, filtering, normalization, weighting, pixel labeling, and post-processing. Therefore, the development of such system consists of both the selection of appropriate techniques among those available in the literature, as well as the development of novel ones. With this in mind, the major contributions and expected outcomes of this research can be summarized as follows:

- The proposed algorithm provides a means to segment the textured regions in image sequences as well as nontextured regions all in one scheme.
- The proposed algorithm does not require high dimensional textured features, as often required for most of the existing texture segmentation algorithms. This is done by incorporating motion feature components into a multiple feature vector, since motion features mainly contribute to the segmentation of textured regions.
- A soft feature weighting scheme is developed, by which only reliable feature components are selected for the segmentation.
- Feature labeling using the SOFM neural networks approach provides fast and parallel operations, since the trained code words can be reused for the segmentation of other image sequences, and the feature labelings are done with vector-matrix multiplications followed by a winner-take-all procedure.
- The proposed algorithm generates more precisely located region boundaries, since the proposed edge fusion subsystem incorporates edge information to produce final segmentation results.

- The extracted feature components can also be directly used for an object description without any further computations, since multiple features such as color, texture, shape (contour of the segmentation), and motion (which are also basic descriptors for MPEG-7) are incorporated into the segmentation process.

- Most existing image sequence segmentation algorithms lack in objective evaluations of their segmentation results. The segmentation results for a variety of MPEG image sequences are evaluated and compared using several evaluation metrics to provide objective evaluation.

- The proposed algorithm is hardware-friendly, since the feature extraction, filtering, weighting, and labeling subtasks are easily mapped into systolic and pipelined architectures. In addition, the process of the edge fusion does not require complex arithmetic.

- A VLSI architecture for the proposed algorithm is proposed, and the computational and hardware complexity of the architecture are analyzed to evaluate the cost and performance tradeoffs. The hardware performance gain of five orders of magnitude over software implementation can be achieved, thus, making it possible to perform real-time image sequence segmentation.

1.5 Organization of the Dissertation

The object-based video processing, representation techniques, and their corresponding standards are briefly reviewed in Chapter 2. It includes some drawbacks of the first generation of video coding in applying the second generation of video coding and content-based object manipulation applications, object-oriented functionalties, MPEG-4 standard encoding scheme and its evaluation methods, ongoing standard, MPEG-7, and MPEG-21.

In Chapter 3, image (sequence) features and typical segmentation methods are discussed. The ill-posed problems of early vision processing, methods for extracting image sequence features such as color, motion, texture, and edges are covered in Chapter 3. In addition, typical image (sequence) segmentation methods and their evaluation metrics are briefly discussed.

Chapter 4 presents the proposed image sequence segmentation algorithm. The previous works and problems of existing image sequence segmentation methods are described in more detail. Also, the feature selection, filtering, and normalization processes are included. The feature weighting scheme suppressing unreliable feature components is developed utilizing motion confidence measures, and the advantages of the feature weighting scheme are discussed for the segmentation of textured image sequences. The multiple feature clustering and labeling method using the SOFM neural networks are also covered in Chapter 4. The edge fusion scheme to provide more meaningful region boundaries of segmentation is developed, and timing analysis, experimental results, and evaluations of the segmentation are included in Chapter 4.

Chapter 5 presents the VLSI architecture of the proposed algorithm. The systolic array architectures for texture feature extraction and motion confidence measure are developed, and motion estimation, edge detection and linking, and feature filtering architecutures are discussed. The pipeline architectures for feature normalization, weighting and labeling (the throughput of these three subtasks is one feature vector per clock) are developed. In addition, the iterative edge fusion scheme is mapped into several subsystems. Also, the complexity of the proposed segmentation algorithm and its VLSI architecture are presented to show the advantages of the dedicated VLSI architecture.

Finally, concluding remarks and suggestion for future work are given in Chapter 6. In Appendices, abbreviations, acronyms, and notations in this dissertation are

attached. In addition, auxiliary experimental data and source codes for both the proposed segmentation algorithm and the VLSI architecture are included in Appendices.

Chapter 2

OBJECT-BASED VIDEO PROCESSING AND REPRESENTATION

2.1 Introduction

The main objective of this chapter is to present the recent developments of object-oriented video processing. The trend is that the scene is processed with the concept of *objects* rather than pixel or block of pixels [9, 20, 21]. The digital representation of an image or a sequence of images requires a very large number of bits. The objective of image coding is to compress the large number of bits as much as possible, and reconstruct an original image or an image sequence faithfully. Waveform-based image coding techniques have been the only approaches utilized in image compression. One of the main problems with these approaches, so-called first generation coding techniques, is that they do not question the image representation structure, since they adapt the concept of pixel or block-based processing.

In order to overcome the limitations of the first generation image coding techniques, the second generation image coding was formally introduced in 1985 [22]. In addition to improved coding efficiency, object-based video processing allows a number of functionalities (see details in Section 2.3.1) which provide solutions to emerging

digital services. For more details of the object-based video coding, the readers should refer to [5].

The first generation video coding schemes are presented in Section 2.2. In Section 2.3, the second generation video coding scheme based on the entities of object is reviewed. The brief introductions of MPEG-4, MPEG-7 and MPEG-21 standards are provided in Section 2.4, 2.5, and 2.6.

2.2 First Generation of Video Coding

The first generation image and video coding techniques achieve compression by reducing the statistical redundancy of image data. First generation coding is based on the theory of rate distortion optimization, which aims at minimizing the distortion given the coding rate. The basic schemes that are generally considered as first generation are pulse code modulation, predictive coding, transform coding, vector quantization and the subband coding and wavelets transform coding.

International standards, MPEG-1 [23] (storage and retrieval of moving pictures and audio on storage media), MPEG-2 [24] (digital television), H.261 [25] and H.263 [26] (video conference), adopted some of these techniques. These schemes achieve the compression by means of hybrid approaches, in which the temporal correlations is exploited with motion estimation and compensation techniques, whereas decorrelation techniques based on waveform coding (mainly DCT) are used in spatial domain. Temporal correlation is obtained by estimating the displacements that 8×8 or 16×16 pixel blocks in the current frame have undergone with respect to reference frames (in the past and future). Spatial decorrelation at each block is then applied to the residual error after the motion compensation, thus reducing the spatial correlation. However, this method creates some drawbacks as follows:

- The semantic content of the image is not taken into account: the blind division results in block effect when high compression ratios are required.

- The block based motion may have little resemblance to the true motion of the objects in the scene.

- The human visual system (HVS) , the most important part of image processing system, is taken into little account. Subband decomposition and wavelets are related to the frequency characteristics of the HVS. But they are based on the pixel as the basic processing element, whereas the HVS interprets the scene based on the image features such as edges, contours and textures (contents of object such as gray-level and color) rather than pixels [22, 27].

The first generation schemes are excellent texture coding schemes. They have proven to provide very good image quality for moderate compression rates. However, the main drawback of these techniques is their lack of a mechanism to cope with the functionalities that allow object-based manipulation and access for the emerging digital services, since the coding schemes are based on the pixels or blocks.

2.3 Second Generation of Video Coding

One of the key issues for the second generation of video coding which was initiated in [22] was the consideration of HVS [27]. The most important conclusions of the study are as follows:

- The edge and contour information are very much appreciated by the HVS and are responsible for our perception.

- Texture is associated with additional information, which is less important than the edges and contours.

- The images of many natural objects can be approximated by members of a class of deterministically self-similar sets.

These conclusions lead to the integration of HVS into the second generation coding system. The most evident result of this trend is the video coding systems based on *segmentation*. The object-based representation can create the new functionalities which the first coding schemes cannot do.

2.3.1 Object-Oriented Functionalities

The introduction of arbitrary objects in the second generation coding scheme also results in a quite remarkable improvement in terms of the functionalities which are required for the new video processing schemes. The analysis of image sequences in term of homogeneous (semantic) regions is indeed a key issue for emerging multimedia services. As far as coding is concerned, the object based manipulation of video will enable a number of functionalities, such as:

Content based functionalities: The different semantic objects can be multiplexed separately in the bitstream, allowing the receiver to manipulate each object independently, combining them according to a composition script that can be transmitted together with the data or generated locally. In addition to this, the decoder can choose to download a subset of the objects, still obtaining a semantically meaningful scene (object scalability).

Improved coding efficiency: The possibility of choosing the best coding strategy independently for each of the objects can significantly improve the coding efficiency. For instance, in a video conference system, the area of the image corresponding to a projected black-and-white transparency can be coded using

bi-level technique, rather than a hybrid coding scheme. Furthermore, the parameters of the coding process itself (quantization step) can be adapted to the characteristics of the object.

Robustness in Error Prone Environments: Part of the bitstream corresponding to different objects can be protected with different levels of error resilence, both at the source and at the channel coding level, according to their relative importance to the overall scene. This will guarantee a higher level of subjective quality in error-prone environments.

Scalability: The object based structure of the bitstream allows two kinds of scalability, namely temporal and spatial scalability. The bitstream can be structured so as to allow the decoder to retrieve the same object at different levels of spatial and/or temporal resolution.

Content description: The possibility of accessing the semantic contents of a scene allows the description of the scene content in a much more efficient way, allowing faster access and retrieval of the desired information.

2.3.2 Segmentation-based Video Coding

The image coding scheme proposed in [22], which took into account the properties of HVS, was based on a contour and texture representation of the image. As far as the still image coding is concerned, the approach is quite simple. Once a good segmentation of the image in homogeneous regions is found, the resulting contours and textures have to be coded. This approach is referred to as segmentation-based image coding. When it comes to the segmentation-based coding of image sequences, the presence of motion poses a number of questions that complicate the scenario:

- In addition to the segmentation in the spatial domain, segmentation in the temporal domain can be applied, either alone or in association with the spatial information (spatio-temporal segmentation).

- The concept of contour in the still image has to be extended in the temporal dimension, together with the constraints on its temporal coherence and continuity. Indeed, region tracking becomes therefore mandatory for the content-based functionalities.

- The motion estimation techniques used in first generation coding scheme may be not efficient when applied in a framework where objects are addressed, rather than pixels or blocks.

The main characteristic of the segmentation-based coding schemes is that the partition of the scene is signal dependent and not determined a priori as it is the case for block based first generation schemes. These techniques therefore rely on two stages, namely an analysis step (mainly segmentation) and a synthesis step (mainly the coding functions).

The analysis and the synthesis phases are performed sequentially and independently [5]. In this scheme, images of the incoming sequence are decomposed into arbitrarily shaped moving objects. Each moving object is described by three sets of parameters defining its shape, motion and color. While coding of color parameters requires a relatively high amount of bits, coding of shape and motion parameters does not. Therefore, in a first step, the current image is synthesized using only the shape and motion parameters of the current image and the motion compensated color parameters of the previous image. Then, for each object those areas are detected where this synthesis does not lead to a sufficient quality. For these model failure (MF) areas, the frame to frame changes cannot be described by the underlying source model. For

all objects, motion and shape parameters are coded and transmitted to the receiver. In order to reach a high coding efficiency, color parameters are coded and transmitted only for the MF areas of the objects.

2.4 MPEG-4: Standard for Multimedia Applications

MPEG-4 is an ISO/IEC standard developed by MPEG (Moving Picture Experts Group), the committee that also developed the Emmy Award winning standards known as MPEG-1 and MPEG-2. These standards made interactive video on CD-ROM and Digital Television possible. MPEG-4 is the result of another international effort involving hundreds of researchers and engineers from all over the world. MPEG-4, whose formal ISO/IEC designation is ISO/IEC 14496, was finalized in October 1998 and became an International Standard in the first months of 1999. Currently, MPEG is finalizing backward compatible extensions under the title of MPEG-4 Version 2.

MPEG-4 builds on the proven success of three fields: digital television, interactive graphics applications (synthetic content), interactive multimedia (World Wide Web, distribution of and access to content). MPEG-4 provides the standardized technological elements enabling the integration of the production, distribution and content access paradigms of the three fields.

The main characteristic of MPEG-4 is that it will be the first coding standard in which data is considered as a composition of separately coded objects, allowing the independent access and manipulation of each object. The key concept in the standard is the Audio Visual Object (AVO), which is defined as a representation of a real or virtual object that can be manifested aurally and/or visually. AVOs are generally hierarchical, in that they may be defined as composites of other AVOs called sub-objects. For a simple example, an AVO inserted into a scene like news broadcasting

is an application that has so far been addressed in TV studios by means of analog techniques known as chroma keying.

The MPEG-4 standard provides a set of technologies to satisfy the needs of authors, service providers and end users alike.

- For authors, MPEG-4 enables the production of content that has far greater reusability, has greater flexibility than is possible today with individual technologies such as digital television, animated graphics, World Wide Web (WWW) pages and their extensions. Also, it is now possible to better manage and protect content owner rights.

- For network service providers, MPEG-4 offers transparent information, which can be interpreted and translated into the appropriate native signaling messages of each network with the help of relevant standards bodies. The foregoing, however, excludes Quality of Service (QoS) considerations, for which MPEG-4 provides a generic QoS descriptor for different MPEG-4 media. The exact translations from the QoS parameters set for each media to the network QoS are beyond the scope of MPEG-4 and are left to network providers. Signaling of the MPEG-4 media QoS descriptors end-to-end enables transport optimization in heterogeneous networks.

- For end users, MPEG-4 brings higher levels of interaction with content, within the limits set by the author. It also brings multimedia to new networks, including those employing relatively low bitrate, and mobile ones. An MPEG-4 applications document exists on the MPEG Home page (www.cselt.it/mpeg), which describes many end user applications, including interactive multimedia, broadcast and mobile communications.

MPEG-4 achieves these goals by providing standardized ways to:

- Represent AVOs, which can be of natural or synthetic origin. This means they could be recorded with a camera or microphone, or generated with a computer.
- Describe the composition of these objects to create compound media objects that form audiovisual scenes.
- Multiplex and synchronize the data associated with media objects, so that they can be transported over network channels providing a QoS appropriate for the nature of the specific media objects;
- Interact with the audiovisual scene generated at the receivers end.

These goals are achieved by the two basic elements of the standard, i.e

- A set of coding tools for audio visual objects, capable of providing support to different functionalities, such as object-based interactivity and scalability, and error robustness, in addition to efficient compression.
- A syntactic description of coded audio visual objects, providing a formal method for describing the coded representation of these objects and the methods used to code them.

2.4.1 Video Encoding

The concept of Video Objects (VO) has been introduced in MPEG-4 to allow content/object level interactivity. A VO can be a specific object, such as a person, car, building, etc. or it can be a broad object such as the foreground, background, region, etc. A Video Object Plane (VOP) is an instance of a VO. For natural video, a VOP is a 2D representation of an object at a given time.

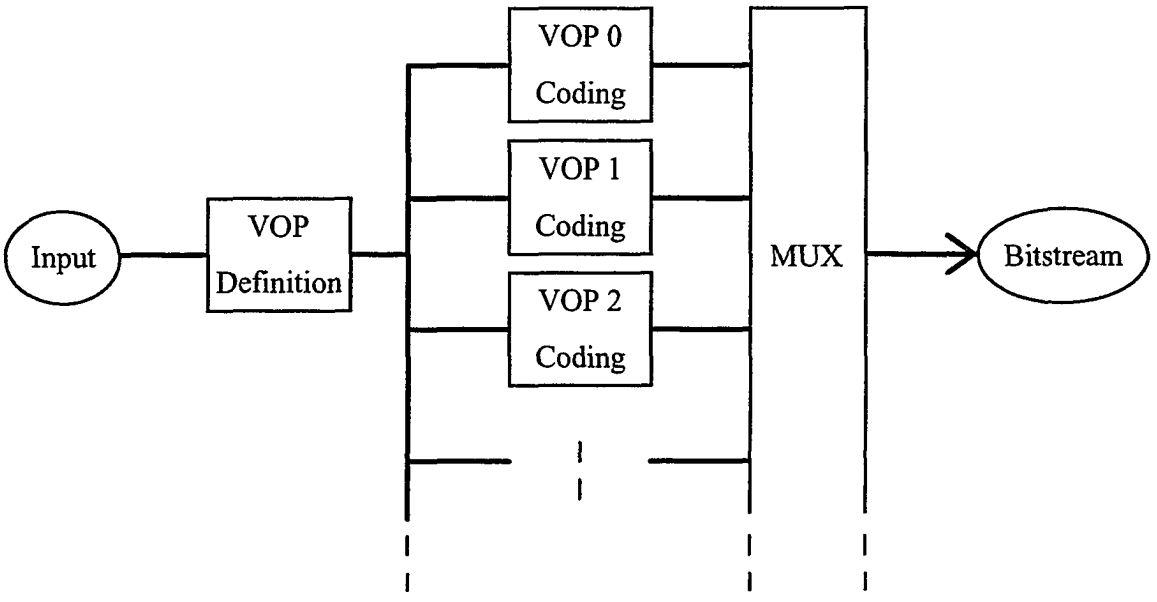
MPEG-4 has introduced the concept of Video Object Layers (VOL) to facilitate object based temporal and spatial scalability. In nonscalable video bitstreams, there

is only one layer per object, the base layer. In scalable video bitstreams, there are two or more layers per object. The first layer is the base layer. It is decoded independently of the other layers. The other layers are known as the enhancement layer(s). Each of the enhancement layers is decoded with respect to the layer immediately below it. Figure 2.1 depicts the general MPEG-4 video encoding scheme and the VOP encoder structure.

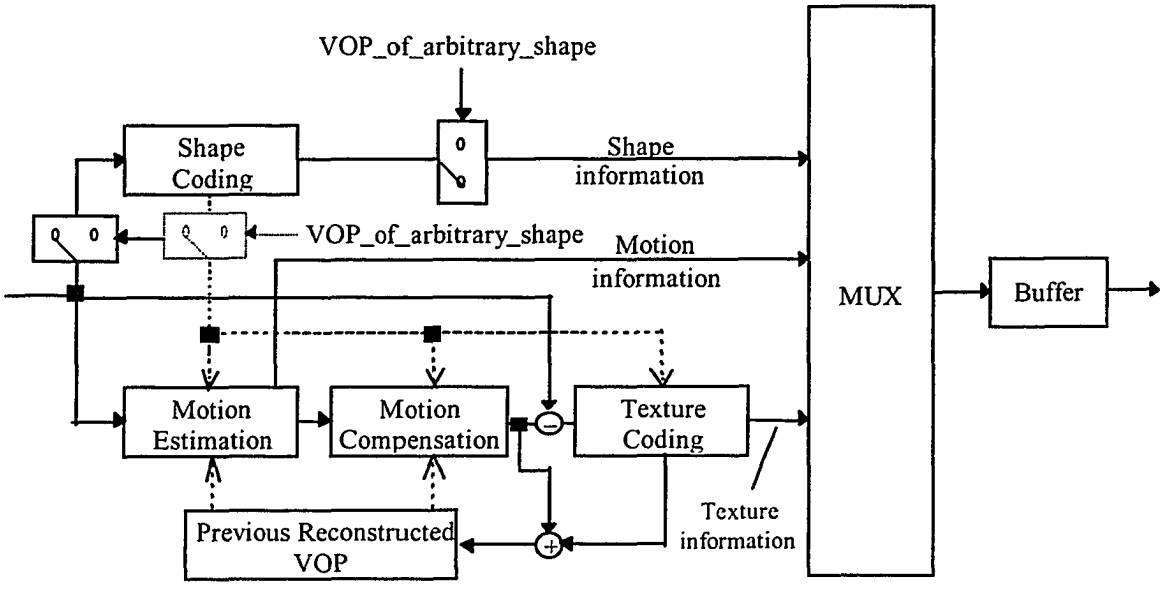
At the encoder, a stream of input pictures is fed into the VOP definition block where each scene is decomposed into its component VOPs. Each of the VOPs is coded individually and is multiplexed together into coded bitstream. *MPEG-4 does not specify the method used to define the VOPs. VOPs can be defined automatically, semi-automatically (with human assistance) or completely manually.* Each VO and VOP can be accessed and manipulated in the coded bitstream. Therefore, by making VOPs semantically meaningful, the specification allows for meaningful interaction with scene content. Each VOP can be arbitrarily shaped.

MPEG-4 has introduced the concept of alpha channels (or shape information) to assist in the composition process. Each pixel in a VOP has an alpha channel associated with it. The bounding rectangle is then extended to multiples of 16×16 blocks to form an extended alpha plane. The extended alpha samples are set to zero. The extended alpha planes are then broken into blocks of 16×16 (known as alpha blocks). The shape encoding process is carried out on the alpha blocks.

Motion compensation is performed on macroblocks (16×16 blocks). Motion-estimation and compensation can also be performed on 8×8 blocks if the overhead required to represent motion information can be tolerated. MPEG-4 features an improved motion compensated prediction technique. Polygon matching, rather than block matching, is used to estimate the motion of the blocks on the boundary of the arbitrarily shaped VOP.



(a) MPEG-4 encoder structure



(b) Video VM VOP encoder structure

Figure 2.1: The MPEG-4 Video Encoder structure, Source:[3].

Intra VOPs and the residual data after motion compensation are coded using a 8×8 block DCT scheme. The DCT is performed on the luminance and the chrominance planes separately. For arbitrary shaped VOPs where the intra 8×8 macroblock lies on the border of the VOP shape, the blocks are first padded and the blocks are then coded using the coding technique used in the H.263.

In order to leave space for competition among different implementations and give flexibility to the MPEG-4 codec, the method to obtain the AVOs are not included in the standard. Similarly, the encoder itself is not standardized, as compliance with the standard is requested only at the bitstream level. The coding tools for each of the objects can be chosen dynamically according to the characteristics of the object itself.

2.4.2 Evaluation of MPEG-4 Video Codec

MPEG carries out verification tests to check whether the standard delivers what it promises [28]. In order for the MPEG-4 subjective tests to be fair and effective, traditional subjective test methods had to be adapted such that the tests evaluate new coding functionalities, low target bitrates video and a large number of different types of impairments [28]. Four test methods are used in the MPEG-4 video subjective tests: double stimulus continuous quality scale, double stimulus impairment scale, double stimulus binary vote, and single stimulus.

2.4.2.1 Double Stimulus Continuous Quality Scale (DSCQS) Method

In the DSCQS method, each trial consists of a pair of stimuli: one is the reference, and the other is the test. The test stimulus is usually the reference after undergoing some type of processing. The two stimuli are each presented in a trial, in alternating fashion, with the order of the two randomly chosen for each trial.

Test subjects are not informed of the ordering of the test and reference stimuli, and they rate each stimulus by marking a continuous quality scale. Occasionally in a

DSCQS test, both stimuli presented in a trial are the reference stimulus. Such trails are used to detect erratic test subject behavior.

This method is typically applied for evaluations where the quality difference between the test and reference sequences is not too large. In MPEG-4 tests, this method is applied to the highest bit rates (320 $\sim$ 1024 kbit/s).

2.4.2.2 Double Stimulus Impairment Scale (DSIS) Method

As in the DSCQS method, each trial consists of a pair of stimuli. However, the two stimuli are always presented in the same order: the reference is always first, followed by the test. In the DSIS method, test subjects compare the two stimuli in a trial and rate the impairment of the test stimulus with respect to the reference using a five-level degradation scale such as imperceptible, perceptible but not annoying, slightly annoying, annoying and very annoying.

This method is suitable for evaluating the performance of systems that introduce clearly visible impairment at low and medium bit rates (48 $\sim$ 320 kbit/s).

2.4.2.3 Double Stimulus Binary Vote Method (DSBV) Method

The DSBV method is a nonstandard method, specially designed by MPEG test experts to evaluate the performance of a codec following a very long bust of bit error causing loss of codec synchronization. Very bad decoded video quality is expected under these conditions. Therefore, test subjects are asked to evaluate whether the codecs under test recover from the error burst by the end of the test sequence, and vote "yes" or "no'.

2.4.2.4 Single Stimulus (SS) Method

Only one stimulus is presented in the SS method. The test subject rates each stimulus, typically using a five-level quality scale (i.e., Excellent, Good, Fair, Poor,

Bad). The SS method is appropriate when references are not available. At lowest tested bit rates, appropriate reference sequences are not readily available. Thus, the SS test method was applied for evaluating the performance of proposals at very low bit rates ($10 \sim 48$ kbit/s).

During the MPEG-4 standardization, it became clear that not only was it important to provide users with efficient ways of coding and manipulation AVOs, but it was important to devise appropriate methods to access and organize huge amounts of data. A new standard, known as MPEG-7, is currently under development.

2.5 MPEG-7: Multimedia Content Description Interface

Nowadays, more and more audiovisual information is available, from many sources around the world. Also, there are people who want to use this audiovisual information for various purposes. However, before the information can be used, it must be located. At the same time, the increasing availability of potentially interesting material makes this search more difficult. This challenging situation led to the need of a solution for the problem of quickly and efficiently searching for various types of multimedia material interesting to the user. MPEG-7 wants to answer to this need, providing this solution.

MPEG-7, formally named "Multimedia Content Description Interface", aims to create a standard for describing the multimedia content data that will support some degree of interpretation of the informations meaning, which can be passed onto, or accessed by, a device or a computer code. MPEG-7 is not aimed at any one application in particular; rather, the elements that MPEG-7 standardises shall support as broad a range of applications as possible. MPEG-7 will be a standardized description of various types of multimedia information. This description will be associated with the

content itself, to allow fast and efficient searching for material that is of interest to the user. MPEG-7 is in the phase of defining committe draft and international standard for MPEG-7 will be issued in September 20001. MPEG-7 will standardize: i) a set of description schemes, ii) a language to specify description schemes (and possibly descriptors), iii) the Description Definition Language (DDL), iv) one or more ways to encode descriptions.

For more details regarding the MPEG-7 background, goals, areas of interest, requirements, development process and work plan, please refer to "MPEG-7: Context, Objectives, and Technical Roadmap", "Applications for MPEG-7", "MPEG-7 Requirements", and "MPEG-7 Development Process". This framework opens new perspectives to the field of image analysis. The challenge is not only to extract information from visual scenes in order to produce objects in view of MPEG-4 coding, but also to extract relevant features that could help in describing each object as well as the complete scene.

2.6 MPEG-21: Multimedia Framework

Nowadays, many elements exist to build an infrastructure for the delivery and consumption of multimedia content. There is, however, no big picture to describe how the specification of these elements, either in existence or under development, relate to each other. The aim of starting MPEG-21 has been: i) to understand if and how these various components fit together and ii) to discuss which new standards may be required, if gaps in the infrastructure exist and, once the above two points have been reached, iii) to actually accomplish the integration of different standards [29].

The digital market place, which is founded upon ubiquitous international communication networks such as the Internet, rewrites existing business models for trading

physical goods with new models for distributing and trading digital content electronically. In this new market place, it is becoming increasingly difficult to separate the different intellectual property rights which are associated with multimedia content. The quest to bring the ultimate experience in multimedia entertainment to the consumer means that the boundaries between the delivery of audio sound (music and spoken word), accompanying artwork (graphics), text (lyrics), video (visual) and synthetic spaces will become increasingly blurred. New, complex solutions are required to manage the delivery process of these different content types in an integrated and harmonized way, entirely transparent to the consumer of multimedia services. And this is only one of the issues that needs to be addressed; there are others, like finding content and ensuring quality of service.

A multimedia framework is required to support this new type of commerce. Such a framework requires that a shared vision, or roadmap, is understood by its architects, to ensure that the systems that deliver e-content are interoperable and that transactions are simplified and, if possible, automated. This should apply to the infrastructure requirements for content delivery, content security, rights management, secure payment, and the technologies enabling them and the list is probably not exhaustive. The scope of MPEG-21 could therefore be described as the integration of two critical technologies: how consumers can search for and obtain content either personally, or through the use of intelligent agents and how content can be presented for consumption according to usage rights associated with the content. Work on the new standard, MPEG-21 has just started in June 2000.

Chapter 3

IMAGE FEATURES AND SEGMENTATION METHODS

3.1 Introduction

For segmentation of an image or an image sequence, image features consisting of homogeneity criteria are extracted. Image features such as edge and motion face ill-posed problems as discussed in Section 3.2. In Section 3.3, image features such as color coordinate systems, texture features, motion, and edge detection methods are presented. In addition, typical image and image sequence segmentation techniques are described in Section 3.4 and 3.5, respectively. Finally, segmentation evaluation metrics are discussed in Section 3.6.

3.2 Ill-posed Problems of Early Vision Process

Early vision consists of a set of process that recover physical properties of the visible three-dimensional surfaces from the two-dimensional images [30]. Examples of early vision processes are edge detection, computation of optical flow, shape from contours, shape from texture, structure from stereo and segmentation etc.

A good definition of early vision is that it is inverse optics. In classical optics or in computer graphics, the basic problem is to determine the images of three-dimensional objects, where vision is confronted with the inverse problem of recovering

surfaces from images. As so much information is lost during the imaging process that projects the three-dimensional world into two-dimensional images, vision must often rely on natural constraints, that is, assumptions about the physical world, to derive unambiguous output.

Two important problems in early vision are the computation of motion and edge detection. The computation of two-dimensional field of velocities in the image is a critical step for recovering the motion and the three-dimensional structure of objects. Local motion measurements provide only the normal component of velocity. The tangential component remains invisible to purely local measurements. The problem of estimating the full velocity field is thus, in general, underdetermined by the measurements that are directly available from the image.

Edge detection is the process to detect and localize sharp changes in image intensity. This is a problem of numerical differentiation of image data, which is distorted by the noise unavoidable during the imaging and the sampling processes. Differentiation amplifies noise and this process is thus inherently unstable. Most problems in early vision present similar difficulties.

A problem is well-posed when its solution exists, is unique and depends continuously on the initial data. Ill-posed problems fail to satisfy one or more of these criteria. Note that the third condition does not imply that the solution is robust against noise in practice. For this, the problem must not only be well-posed but also be well conditioned to ensure numerical stability.

Computation of the optical flow is ill-posed because the inverse problem of recovering the full velocity field from its normal component along a contour fails to satisfy the uniqueness condition. Edge detection, intended as numerical differentiation, is ill-posed because the solution does not depend continuously on the data. Segmentation of the image is also ill-posed in the sense that there is no unique solution,

and boundaries of segmented regions lie on pixels where there are abrupt changes of gray/color values.

The main idea for solving ill-posed problems, that is for restoring well-posedness, is to restrict the class of admissible solutions by introducing suitable *a priori* knowledge. *A priori* knowledge can be exploited, for example, under the form of either variational principles that impose constraints on the possible solutions or as statistical properties of the solution space.

3.3 Image Features

3.3.1 Color Space

With the rapid increases of computational power over the last decade, the use of color features, which convey much more information, has received much attention for segmentation [31]. The visual information caused by colored light is richer than that caused by achromatic light. Hue (dominant wavelength) distinguishes among colors such as red, green and yellow. Saturation (excitation purity) refers to how far color is from a gray of equal intensity, white color. Lightness embodies the achromatic notion of perceived intensity of a reflecting object.

According to the tristimulus theory [32], color can be represented by three components, resulting from three separate color filter S_x, $x = R, G, B$, on light radiance $E(\lambda)$ as follows:

$$R = \int_\lambda E(\lambda) S_R(\lambda) d\lambda, \quad G = \int_\lambda E(\lambda) S_G(\lambda) d\lambda, \quad B = \int_\lambda E(\lambda) S_B(\lambda) d\lambda. \tag{3.1}$$

If variations of intensities are uniform across the spectrum, then normalized RGB space is of value:

$$r = \frac{R}{R+G+B}, \quad g = \frac{G}{R+G+B}, \quad b = \frac{B}{R+G+B}. \tag{3.2}$$

RGB color model is used in color input and output devices such as color CRT monitors and video camera. However, RGB is not very efficient for representing real-world images, since equal bandwidths are required to describe all three color components.

There are many color models which are linear or nonlinear transformations of these three primaries. Generally, there is no color system which is good for all color images. Color coordinate systems are specific to the applications. RGB tristimulus coordinates exclude some visible colors. They are depend on physical sensors. For this reason, an international organization responsible for color metrics, the CIE, fixed XYZ coordinates. They can be produced from RGB space by a linear transformation which was determined empirically.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.067 & 0.174 & 0.200 \\ 0.299 & 0.587 & 0.114 \\ 0.000 & 0.066 & 1.116 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \tag{3.3}$$

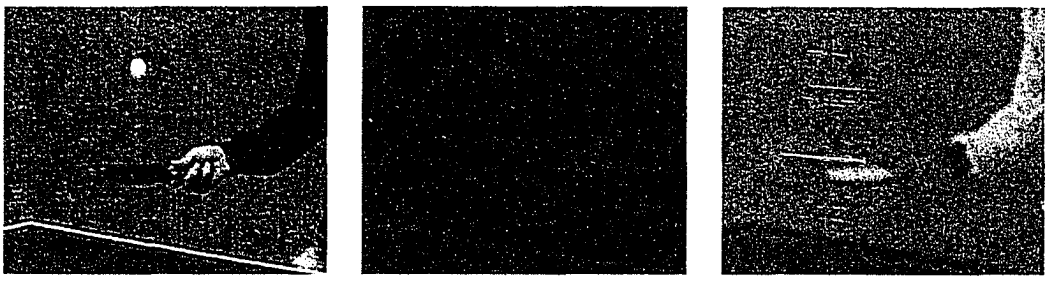
YIQ is used in American TV system. YIQ is a recoding of RGB for transmission efficiency and for downward compatibility with black-and-white television.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \tag{3.4}$$

Note that luminance Y reflects the relative importance of green and red and the relative unimportance of blue which corresponds the luminance efficient function, the eye's response to light of the constant luminance, as the dominant wavelength is varied: our peak sensitivity is to yellow-green light of wavelength around $550nm$. The chromaticity is encoded in I and Q.

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.437 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \tag{3.5}$$

YUV is the European TV system. Y represents luminance and color information is encoded as U and V which are blue and red color difference signals from the Y



(a) Y component. (b) U component. (c) V component.

Figure 3.1: The YUV components of *table tennis frame* 1.

respectively. U and V are subsampled by a factor of two to four, since the human visual system is much less sensitive to chromaticity than the luminance Y.

Figure 3.1 shows the YUV component of *table tennis* frame 1 which consists of a red racket and sleeve, brown background, a white ball, a white hand and a green table. Note that V components have higher value in racket and sleeve. In this dissertation, the MPEG-4 test color image sequences are encoded as YUV format. YUV color coordinate system is used without any kinds of transformation of color system.

3.3.2 Edge

Edge points can be thought of as pixel locations of abrupt changes in scene. Edges usually lie on object boundaries, and therefore very useful for segmentation. The edges can be detected locally based on the point under consideration and some of its neighboring points by using gradient or Laplacian operators. There are different type of gradient operators, such as Roberts, Prewitt and Sobel [33, 34]. Equation 3.6 shows the Sobel operators which possess greater noise immunity. These operators compute the horizontal (O_h) and vertical (O_v) differences using local sums. The pixel location (x, y) is considered as an edge if the gradient magnitude exceeds some

threshold.

$$O_h = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad O_v = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}. \tag{3.6}$$

The Laplacian, on the other hand, is a second difference operator. The Laplacian operator is given by

$$\nabla^2 = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}. \tag{3.7}$$

The discrete Laplacian being a second difference operator, has a zero response to linear ramps. It responds strongly to corners, lines and isolated points. Discrete difference approximations can be used to estimate the second order derivatives. The 3×3 Laplacian convolution mask is shown in Equation 3.8.

$$O_L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}. \tag{3.8}$$

A good edge detector should be a differential operator, taking either the first or second spatial derivative of the image. Second, it should be capable of being tuned to act at any desired scale, so that large filters can be used to detect blurred shadow edges, and small ones to detect sharply focused fine details. The most satisfactory operator fulfilling these condition is the Laplacian of Gausssian operator. It is normally denoted by $\nabla^2 G$.

$$G = e^{(x^2+y^2)/(2\pi\sigma^2)}. \tag{3.9}$$

G is a two-dimensional Gaussian distribution, with standard deviation σ. The shape of distribution can control the amount of smoothing by varying σ.

According to Canny [35], good detectors should have the following three properties: (1) low probability of wrongly marking nonedge points and low probability of failing to mark real edge points (i.e., good detection), (2) points marked as edges should be as close as possible to the center of true edges (i.e., good localization), and

(3) one and only one response to a single edge point (i.e., single response). To maximize simultaneously both good detection and localization criteria, Canny maximized the product of SNR and the reciprocal of standard deviation of the displacement of edge points using a constraint which eliminates multiple responses to single edge points.

There have been attempts to map some of the existing algorithms onto silicon. An ASIC architecture for implementing Sobel operator was proposed in [36]. Another real-time edge detectors and edge linking architecture were proposed in [37, 38, 39].

3.3.3 Texture Features

There are many texture feature extraction methods in the literature. Some of important texture features are described in this section. For reviewing the texture segmentation techniques in more details, please refer to [40].

Small center-weighted filter masks (for example, 5) are convolved with the image to be segmented [41]. Statistics, such as the variance, edge and spot, are computed within a window about each pixel in the resulting images. The values of the statistics are assigned as features to be corresponding pixel in the original image. It is noted that Laws's basic filter functions have a striking similarity to Gabor functions. However, since the filter masks are relatively small, only the high frequency content of textures can be analyzed.

One of the first methods used in the texture segmentation, and still a major one, is the spatial gray level dependence method [42]. The gray level co-occurrence matrix (GLCM), also called gray tone spatial dependence matrix is defined as follow. Given an image $f(x, y)$ with a set of N_g discrete gray levels, define the matrix $P(i, j, d, \theta)$ such that the (i, j)th entry is the number of times that

$$f(x_1, y_1) = i, \quad f(x_2, y_2) = j, \quad where \quad (x_2, y_2) = (x_1, y_1) + (d\cos\theta, d\sin\theta). \quad (3.10)$$

This yields a matrix of dimension equal to the number of gray levels in the image for each distance and orientation (d, θ). This method has proven quite successful. However, many of the texture features have little correlation with features visible to human eye and the many computations are needed.

Statistical methods like GLCM have in the past proven superior to frequency domain techniques. This is due to the lack of locality in these early frequency analysis methods. Specially, there is a large and growing body of theory postulating local frequency analysis in the human visual system. In particular, models of the human vision system that use Gabor filters to model the receptive fields sufficiently account for psychophysical data obtained in texture discrimination experiments. Texture analyzers implemented using $2D$ Gabor functions have produced a strong correlation between outputs of banks of $2D$ Gabor filters with actual human segmentation. A Gabor elementary function [43] is given by

$$h(x, y) = g(x', y') \exp j2\pi(U_x + V_y), \tag{3.11}$$

where $(x', y') = (x\cos\theta + y\sin\theta, -x\sin\theta + y\cos\theta)$ represents rotated spatial-domain rectilinear coordinates, (U, V) represents a particular $2D$ frequency. The complex exponential is a $2D$ complex sinusoid at frequency $F = \sqrt{U^2 + V^2}$. The function $g(x, y)$ is the $2D$ Gaussian

$$g(x, y) = \frac{1}{2\pi\sigma_x\sigma_y} \exp[-\frac{1}{2}(\frac{x^2}{\sigma_x} + \frac{y^2}{\sigma_y})], \tag{3.12}$$

where σ_x and σ_y characterize the spatial extent and bandwidth of the filter.

From the frequency response of the Gabor filter, it has the shape of a Gaussian. The Gaussian's major and minor axis widths are determined by σ_x and σ_y. It is rotated by an angle θ with respect to the positive u-axis, and it is centered about the frequency (U, V). Thus, the Gabor filter acts as a bandpass filter.

Gabor functions are used for texture segmentation by finding the filters that are tuned to the image's dominant spectral information. For this, we should select parameters for frequency, orientation and bandwith for each filter. Each point in the images is labeled according to which filter output gives highest energy value for the points. The result is that the image is segmented into regions according dominant spectral information. There are drawbacks with using the Gabor filters in practical application as follows:

- The selection of filters (parameters) is data dependent and nontrivial.
- The application of the filters to images is not simple. The computation of the filter coefficients is a complex process because the Gabor functions are not orthogonal.
- Discrete versions of the Gabor function must be obtained in order to be applied to images.

Another way is with a dyadic filter bank. This produces octave bandwith segments in spatial-frequency. It simultaneously allows for high spatial resolution at high spatial-frequencies and high spatial-frequency resolution at low spatial-frequencies.

A very practical filter bank for image decomposition is the quadrature mirror filter (QMF) bank [44]. It provides a low cost decomposition of the image. Several aspects of the QMF filter bank as being relevant to texture extraction can be identified: (1) ability to achieve perfect reconstruction, (2) outputs are localized filters and the decimation of filter outputs reduces complexity, and (3) QMF features performed better than those proposed by Haralick (co-occurrence matrices) [42]. One of applications of wavelet feature for texture extraction is as follows.

- The value at image points in each filter output is replaced with the absolute value of the output.

- Measure the band limited spatial-frequency energy at each image point.
- Each of the filter outputs is thresholded to produce a binary image. Each binary image will have values at points of high energy that correspond to the presence of lines, edges, noise and textures within the original image.
- Apply nonlinear filtering (median filter) to reduces the edge, line and noise pattern (smoothing) and remove regions that are below a size threshold.
- The full image is reconstructed such that each image point is replaced by a binary vector.

3.3.4 Motion

Observed motion is not the true motion, but apparent motion induced by temporal changes of two dimensional image intensity. For image sequence segmentation purpose, it is natural to include motion information among the criteria used for the segmentation itself.

$2D$ motion estimation, based only on two frames, is an ill-posed problem [45] in the absence of any additional assumption about the nature of the motion. $2D$ motion estimation suffers from all of the existence, uniqueness and continuity problems [4]:

- No correspondence can be established for covered/uncovered background points. This is known as the occlusion problem.
- If the components of the displacement at each pixel are treated as independent variables, then the number of unknowns is twice the number of observations (the elements of the frame difference). This is called aperture problem.
- Motion estimation is highly sensitive to the presence of observation noise in video images. A small amount of noise may result in a large deviation in the motion estimates.

In general, motion estimation methods are classified into two categories: optic flow estimation techniques based on the optic flow constraint and region based matching techniques. The use of the Laplacian allows the computation of large displacements between frames. Lucas and Kanade [46] estimated optic flows by adding a spatial neighborhood function to the first order optic flow constraint to overcome the aperture problem. Anandan [47] estimated dense motion based on a Laplacian pyramid and a coarse-to-fine sum of squared difference (SSD) based matching strategy. Brief descriptions of these methods are provided in the following subsections.

3.3.4.1 Lucas and Kanade's Optic Flow Estimate

Lucas and Kanade [46] motion model is based on the optical flow contraints that the intensity remains constant along a motion trajectory. The first instances used first order derivatives and were based on image translation.

$$I(\mathbf{x}, t) = I(\mathbf{x} - \mathbf{v}t, 0), \tag{3.13}$$

where $\mathbf{v} = (u, v)^T$. From Taylor expansion of Equation 3.13, optic flow contraints, $dI(\mathbf{x}, t)/dt = 0$, are derived:

$$\nabla I(\mathbf{x}, t) \cdot \mathbf{v} + I_t(\mathbf{x}, t) = 0, \tag{3.14}$$

where $I_t(\mathbf{x}, t)$ denotes the partial time derivative of $I(\mathbf{x}, t)$, $\nabla I(\mathbf{x}, t) = (I_x(\mathbf{x}, t), I_y(\mathbf{x}, t))^T$, and $\nabla I \cdot \mathbf{v}$ denotes the dot product. Equation 3.14 yields the normal component of motion of spatial contours of constant intensity, $\mathbf{v}_n = s$. The normal velocity s and the normal direction n are given by

$$s(\mathbf{x}, t) = \frac{-I_t(\mathbf{x}, t)}{\|\nabla I(\mathbf{x}, t)\|}, \quad n(\mathbf{x}, t) = \frac{\nabla I(\mathbf{x}, t)}{\|\nabla I(\mathbf{x}, t)\|}. \tag{3.15}$$

There are two unknown components of $\mathbf{v}$ in Equation 3.15, constrained by only one linear equation. Further constraints are therefore necessary to solve both components of $\mathbf{v}$.

Lucas and Kanade assumed that the motion vectors are not changed over a particular block of pixels to overcome the aperture problem by introducing small spatial neighborhood Ω by minimizing the following equation in least square sense.

$$\Sigma_{\mathbf{x}\in\Omega} W^2(\mathbf{x})[\nabla I(\mathbf{x},t)\cdot\mathbf{v} + I_t(\mathbf{x},t)]^2, \tag{3.16}$$

where $W(\mathbf{x})$ denotes a window function that gives more influence to constraints at the center of the neighborhood. The solution to Equation 3.16 is given by

$$A^T W^2 A\mathbf{v} = A^T W^2 \mathbf{b}, \tag{3.17}$$

where, for n points $\mathbf{x}_i \in \Omega$ at a single time t, $A = [\nabla I(\mathbf{x_1}),\ldots,\nabla I(\mathbf{x_n})]^T$, $W = diag[W(\mathbf{x_1}),\ldots,\nabla W(\mathbf{x_n})]$, $\mathbf{b} = -[I_t(\mathbf{x}_1),\ldots,I_t(\mathbf{x}_n)]^T$. The solution to Equation 3.17 is $\mathbf{v} = [A^T W^2 A]^{-1} A^T W^2 \mathbf{b}$, which is solved in closed form when $[A^T W^2 A]^{-1}$ is nonsingular, since it is 2×2 matrix:

$$A^T W^2 A = \begin{bmatrix} \sum W^2(\mathbf{x}) {I_x}^2(\mathbf{x}) & \sum W^2(\mathbf{x}) I_x(\mathbf{x}) I_y(\mathbf{x}) \\ \sum W^2(\mathbf{x}) I_x(\mathbf{x}) I_y(\mathbf{x}) & \sum W^2(\mathbf{x}) {I_y}^2(\mathbf{x}) \end{bmatrix}, \tag{3.18}$$

where all sums are calculated over points in the neighborhood.

Equations 3.16 can be considered as weighted least squares estimates of $\mathbf{v}$ from estimates of normal velocities $\mathbf{v}_n = s$. Equation 3.16 is equivalent to

$$\Sigma_{\mathbf{x}\in\Omega} W^2(\mathbf{x}) w^2(\mathbf{x})[\mathbf{v}\cdot\mathbf{n}(\mathbf{x}) - s(\mathbf{x})]^2, \tag{3.19}$$

where the coefficients $w^2(\mathbf{x}) = \|\nabla I(\mathbf{x},t)\|$, reflect a confidence in the normal velocity estimates.

3.3.4.2 Anandan's Region-based Matching

Accurate numerical differentiation may be impractical because of noise, and aliasing in the image acquisition process. In these cases, differential approaches may be inappropriate and it is natural to turn to region based matching methods [47]. Such

approaches define velocity v as the shift $\mathbf{d} = (d_x, d_y)$ that yields the best fit between image regions at different times. Finding the best match amounts to a maximizing a similarity measure over $\mathbf{d}$, such as the normalized cross-correlation or minimizing a distance measure, such as the the sum-of-squared difference (SSD):

$$SSD_{1,2}(\mathbf{x};\mathbf{d}) = \sum_{j=-n}^{n}\sum_{i=-n}^{n} W(i,j) \times [I_1(\mathbf{x}+(i,j)) - I_2(\mathbf{x}+\mathbf{d}+(i,j))]^2, \qquad (3.20)$$

where W denotes a discrete $2-D$ window function, and $\mathbf{d} = (d_x, d_y)$ takes on integer values.

Anandan's matching technique is based on a Laplacian pyramid and a coarse-to-fine SSD-based matching strategy. The Laplacian pyramid allows the computation of large displacements between frames. At the coarsest level, displacements are assumed to be 1 pixel per frame or less. SSD minima are first located to pixel accuracy by computing SSD values in a 3×3 search space, using a 5×5 Gaussian for $W(x)$. Subpixel displacements are then computed by finding the minimum of a quadratic approximation to the SSD surface (about the minimum SSD value found with integer displacements $\mathbf{d}$). Anandan used Beaudet operators to estimate the quadratic surface parameters. Confidence measure, c_{min} and c_{max} are derived from the principle curvatures, C_{min} and C_{max} of SSD surface at the minimum:

$$c_{max} = \frac{C_{max}}{k_1 + k_2 S_{min} + k_3 C_{max}}, \qquad c_{min} = \frac{C_{min}}{k_1 + k_2 S_{min} + k_3 C_{min}}, \qquad (3.21)$$

where k_1,k_2 and k_3 are normalization constants, and S_{min} is the SSD value at the minima. Anandan uses $k_1 = 150$, $k_2 = 1$ and $k_3 = 0$.

Anadan also employs a smoothness constraint on the velocity estimates, taking c_{min} and c_{max} into account. Matching and smoothing are performed at each level of the Laplacian pyramid. When moving from coarser to finer levels the initial 3×3 SSD search area is determined by projecting the coarser-level estimate at each pixel

to all pixels in a 4×4 region at the next-finer level so that each pixel at the finer level has 4 initial guesses. The SSD search area is then the union of the 3×3 areas centered at each of the 4 initial displacements.

3.4 Image Segmentation Techniques

3.4.1 Definition of Segmentation

There are several types of images, such as gray/color image (luminance, chrominance), range image (depth) and nuclear magnetic resonance image (MRI) and thermal image, moving image and texture image, etc. Segmentation is a process of partitioning the image into nonintersecting regions such that each region is homogeneous.

The formal definitions of the segmentation problem are slightly different in popular computer vision and image processing textbooks and survey papers [33, 40]. For instance, consider the definition of segmentation in [33].

Let R represent the entire image region. We may view segmentation as a process that partitions R into n subregions, $(R_1, R_2, \ldots, R_n)$, such that

1. $\bigcup_{i=1}^{n} R_i = R$,
2. R_i is a connected region, $i = 1, 2, \ldots, n$,
3. $R_i \bigcap R_j = \varnothing$, for all i and j, $i \neq j$,
4. $P(R_i) = TRUE$ for $i = 1, 2, \ldots, n$, and
5. $P(S_i \bigcup S_j) = FALSE$ for $i \neq j$,

where $P(R_i)$ is a uniformity predicate over the points in set R_i and $\varnothing$ is the null set. Condition (1) indicates that every pixel must be in a region. The second condition requires that points in a region must be connected. Condition (3) indicates that the regions must be disjoint. Condition (4) determines what kind of properties the segmented regions should have, for example, uniform gray levels. Finally, condition (5) indicates that regions R_i and R_j are different in the sense of predicate P.

Condition (5) of this definition must be modified to apply only to adjacent regions, as nonneighboring regions may have the same region label if regions satisfy the same predicate constraints, regardless of whether they are spatially connected. This violates condition (2) of the above definition. For gray level images, the uniformity predicate measures the uniformity of luminance value, whereas for textured images, texture features could be uniformity predicate.

It is very difficult to define image sequence segmentation formally, since a homogeneous predicate is not always motion. Some applications need to segment moving regions using only motion, and the others need to extract a region of nonmoving background using only static features, or a mixture of static (color, texture) and dynamic (motion) predicate. The formal definition of image sequence segmentation has been abandoned in favor of informal rules. The definition of homogeniety is a critical factor for segmentation. Different definition of homogeniety leads to totally different segmentation results for the same input. For example, a single color homogeniety leads to region based segmentation, while a single motion homogeneity creates the motion based segmentation. What is the good homogeneous criterion is application specific, especially for image sequence segmentation.

Hundreds of segmentation methods have been presented, but there is no unified method which can be considered good for all the images. Most algorithms are specific to the image formation model. Algorithms developed for one class of images may not always be applied to the other classes. The segmentation is still a very challenging topic in the computer vision and image processing community: (1) homogeneous criteria are scene-dependent, (2) no ground-truth segmentation information is available, (3) the number of final segmented regions is usually unknown, (4) there is uncertainty problem [48, 49] in the signal processing, and (5) segmentation is an ill-posed problems [12].

3.4.2 Gray Level Thesholding

Thresholding is perhaps the most intuitive form of image segmentation. Histogram analysis is computationally inexpensive cluster analysis techniques. The procedure known as global thresholding simply needs to examine the histogram and to place the threhshold value in the valley [50, 34].

A major drawback of segmentation methods based on thresholding is that they tend to ignore spatial information in the image. In order to overcome the problems, more sophisticated methods have been devised to combine spatial information with the luminance value. A histogram is evaluated for only those pixels having a high magnitude of the Laplacian of the luminance function (selection of the areas where region borders are more likely to be present) [51]. Another approach for the optimization of the threshold selection has been proposed, based on a local measure of activity [52].

In order to reduce the influence of noise and illumination change, local or adaptive thresholding techniques have been proposed [53]. The method can be applied to specific scenes such as bi-level segmentation of simple shaped image. In order to achieve the localization, histograms can be formed over a rectangular decomposition of the image [50]. The sectors, which initially are histogramed and segmented independently, are formed by 16×16 or 32×32 pixel sectors. These initial segmentations are then simplified using a region merging algorithm.

3.4.3 Region Growing/Splitting and Merging

Region growing is one of the conceptually simplest approaches to image segmentation: neighboring pixels of similar characteristics are grouped to form a segmented region [54, 55]. Constraints must be placed on the growth pattern in order to achieve acceptable results. Region growing methods are in principle applicable whenever a

distance between two pixel can be defined. Region growing methods strongly depend on the estimation of the measure of similarity between pixels and the use of connected components. Another drawback of such methods is their inherently sequential nature: the final results depends on the the scanning strategy that has been adopted.

Splitting and merging methods are based on a tree data representation whereby an image region is split into subregions when it is not uniform in one or more features [56, 57]. If neighboring regions are found to be uniform, then they are merged by a single region composed of the adjacent regions. The most common approach to split and merge segmentation is based on a quadtree partition of the image. The initial regions are square and the splitting iteratively partitions the regions into 4 subregions. This choice has the advantage of simplifying the description of the partition tree, but tends to produce blocky segmentation. For the post-processing of oversegmented regions, only region merging process can be applied for them in order to generate the final segmentation results with lower resolution. If $P(R_i \cup R_j)$ is true, any adjacent regions R_i and R_j are merged.

3.4.4 Recursive Shortest Spanning Tree (RSST)

A problem with any method of segmentation is to define the level of detail which is of interest. Sometimes fine detail is important, while in other cases larger-scale features are more significant. This variation in the "scale of interest" means that most methods developed and currently used to date that are suitable for one situation may not be so for another. In order that segmentation techniques be effective in all cases, they must be able to describe regions in a hierarchical order of importance. For example, if two regions are required then the two most significant regions are found. If a third region is desired, a further partition must be made so that the next most significant region is obtained. Consequently, if only a few regions are generated,

global features of the image are represented and, as more regions are generated, more detail in the image signal is uncovered [58].

Minimum spanning tree, a concept equivalent to the shortest spanning tree (SST) can be used for cluster analysis in graphs to produce connected groups of vertices. The spatial Euclidean distance between these vertices was used as the weight function to be minimized by the tree which has been constructed by graph theory. The SST segmentation method has some problems that limit its usefulness: i) no global information is used in finding the SST, and the segmentation of noisy image cannot fare well, ii) regions which are very different may be merged if they are connected by a series of pixels which differ only slightly form each other, but form a large overall difference.

In order to improve the SST segmentation, the recursive SST (RSST) segmentation method uses the global information. As the pixel values are modified according to all the pixels in the regions, it is possible for one pixel to affect another which is not necessarily its nearest neighbor. The link weights between regions will also be altered to take account of all the pixels in both the regions that the link connects, so that global region information is incorporated into the link weights. The recursive SST segmentation algorithm for obtaining M regions is as follows: (1) map the image onto a weighted graph, (2) while there is more than one vertex in the graph, repeat (3)$\sim$(7), (3) find the next least weighted link, (4) save the link, (5) merge the two vertices joined by this link, (6) recalculate the new vertex weight and link weights, (7) remove duplicated links, (8) form a spanning tree with the saved links, (9) cut the spanning tree at the $M-1$ most costly links to form a spanning forest, and (10) map the forest onto a segmentation image.

The RSST segmentation method incorporates global information of the image and the final spanning tree keeps the hierarchy of the segmentation resolution. How-

ever, the RSST method requires the predefined number of segmented regions and a high computational complexity.

3.4.5 Clustering

Clustering techniques are well adapted when the multiple features of image are incorporated in the process. Typically, several image characteristics (features) are extracted from small local areas of the image and constitute a feature space. Each homogeneous region in the feature space is assumed to constitute a cluster. Clustering methods try to group features with some kinds of distance measures, and the resulting clusters are mapped back on the image, generate the segmentation [59, 60, 61, 40, 62].

The clustering in the feature space is based on the following elements. Let's $\mathbf{X} = \mathbf{x}_1, \ldots, \mathbf{x}_n$ be the data set, $M = \mu_1, \ldots, \mu_c$ the set of centroids corresponding to the c classes, $\|\cdot\|^2$ is a distance measure in the feature space and $U = u_{ik}$ is a matrix of belongingness of dimensions $n \times c$ such that: if element $\mathbf{x}_i$ belongs to cluster k, $u_{ik} = 1$ and otherwise, 0 with $\sum_{i=1}^{c} u_{ik} = 1 \ \forall k$ which guarantees that each element belongs to one and only one cluster. A clustering algorithm can be defined as a scheme that evaluates a matrix U minimizing an objective function:

$$J(c, \mathbf{X}, U, M) = \sum \sum u_{ik} \|\mu_i - \mathbf{x}_k\|^2. \tag{3.22}$$

Clustering methods can be determined by the structure of the data set, distance measure and the properties of the belongingness matrix. When the u_{ik} is noninteger value, the matrix can then be defined as:

$$u_{ik} \in [0, 1], \quad \sum_{i=1}^{c} u_{ik} = 1 \quad \forall k, \tag{3.23}$$

where c is the number of clusters. This formulation corresponds to the case of fuzzy clustering methods. The classical clustering algorithm, known as ISODATA (hard C-Means) [60] uses an integer value membership (0 or 1) for the belongingness matrix. In

order to approximate the minima of the objective function, an iterative optimization method are used. During the iterations, the centroids of cluster and the memberships (hard partition, 0 or 1) are updated until there is no big difference in them.

Clustering algorithms do not take into account the spatial information of pixel unless the pixel positions consist of feature vector [63]. The use of statistical models such as Markov Random Field (MRF) has been proposed among other solutions in order to assure a higher degree of spatial correlation to the segmentation. Also clustering methods normally require a priori knowledge of the number of clusters that need to be obtained. It is clear that this decision represents a limitation in that a wrong choice can result in oversegmentation (i.e., the creation of numerous unwanted small regions), or undersegmentation (undesired merging of adjacent but nonhomogeneous regions). This problem is also known as cluster validity problem. Various approaches have been proposed in order to estimate the good number of clusters in segmentation [64]. However, in general the evaluations are done after segmentation.

3.4.6 Statistical Methods

The goal of statistical segmentation method is to evaluate a statistical model of the distribution of the pixels into regions [65, 66]. Parametric estimates are the most popular tools adopted in this framework. The shape of the underlying distribution is assumed to be known and the method aims at estimating the parameters of the distribution. As an example, let us consider the case in which we want to partition an image into r regions, operating in an f-dimensional multivariate space. Assume that the distribution of the feature vectors in each of the regions is an f-variate normal distribution, where μ_i $(1 \leq i \leq r)$ are the mean vectors and Λ_i $(1 \leq i \leq r)$ are the corresponding covariance matrices. The pixels in the image can be thought of

as drawn from a mixture of r populations, according to mixing proportions that we shall indicate with ω_i $(1 \leq i \leq r)$.

The goal of the analysis is determining the set of parameters that characterize the mixture, $\theta = (\omega_1, \ldots, \omega_r, \mu_i, \ldots, \mu_r, \Lambda_i, \ldots, \Lambda_r)$. The most popular method for the estimation of parameters are the Maximum Likelihood (ML) method [65]. The density $p(\mathbf{x}|\theta)$ corresponding to a mixture can be expressed by:

$$p(\mathbf{x}|\theta) = \sum_{i=1}^{c} \omega_i f_i(\mathbf{x}|\mu_i, \Lambda_i). \tag{3.24}$$

If the distributions that contribute to the mixture are f-variate Gaussians, the $f_i(\mathbf{x}, \theta)$ is:

$$f_i(\mathbf{x}, \theta) = (2\pi)^{-\frac{r}{2}} |\Lambda_i|^{-\frac{1}{2}} \exp[-\frac{1}{2}(\mathbf{x} - \mu_i)^t |\Lambda_i|^{-1} (\mathbf{x} - \mu_i)], \tag{3.25}$$

where u_i and Λ_i are the covariance matrices expressed as:

$$\mu_i = E[\mathbf{x}_i]. \tag{3.26}$$

$$\Lambda_i = E[(\mathbf{x} - \mu_i)(\mathbf{x} - \mu_i)^t]. \tag{3.27}$$

Assuming that $\mathbf{x}$ is a set of n independent observations in the feature space, the likelihood function is given by:

$$L_\theta = \Pi_{k=1}^{n} f(\mathbf{x}_k|\theta) = \Pi_{k=1}^{n} \sum_{i=1}^{c} \omega_i f_i(\mathbf{x}, \theta). \tag{3.28}$$

The goal of ML is to find θ that maximizes the likelihood L_θ: find the set of parameters that represent the mixture so well that the probability that it belongs to that mixture is the highest possible.

The Expectation Maximization (EM) algorithm is an iterative solution that find the optimal parameters of a mixture density, for a specific number of mode r. In the first step, parameters are initialized. For each data sample point, calculate the conditional class probability, i.e., the expectation that each sample belongs to each

mode. This step is referred to as expectation or E-step. Then, update the estimates of the parameters. In this way, the maximum likelihood values of the parameter vectors are evaluated. This step is referred as maximization or M-step. The above steps are repeated until the variations of the parameter values remain below a given threshold.

Statistical methods seem promising: these methods evaluate alternatively the localization of the boarders and the characterization of the regions. And they are well adapted to the use of multiple features of image such as luminance, motion and texture. However, the performance of the method depends strongly on the fitness of the assumed model (for example: normal distribution) to the reality of the data. Also their computational complexities are very high, since at each step the covariance matrices and their inverses need to be evaluated.

3.4.7 Markov Random Fields

Markov Random Fields (MRF) have become a popular tool as a priori signal models in Baysian estimation problems that can be applied to texture modeling and generation [67], image segmentation and restoration and motion segmentation [68, 69], and motion estimation [70].

MRF is an extensions of 1-D causal Markov chains to 2-D, and has been traditionally specified in terms of local conditional probability density functions (pdf) which limit their utility [68].

Definition of MRF: The random field $\mathbf{z} = z(\mathbf{x})$, $\mathbf{x} \in \Lambda$ is called a Markov random field with respect to $\mathcal{N}$ if $p(\mathbf{z}) > 0$, for all $\mathbf{z}$ and, $p(z(\mathbf{x}_i)|z(\mathbf{x}_j),\ \forall \mathbf{x}_j \neq \mathbf{x}_i) = p(z(\mathbf{x}_i)|z(\mathbf{x}_j),\ \mathbf{x}_j \in \mathcal{N}_{\mathbf{x}_i}))$, where the pdf of a discrete valued random variable/field is defined in terms of a Dirac delta function.

The first condition states that all possible realizations should have nonzero probability, and the second condition requires that the local conditional pdf at a particular site

$\mathbf{x}_i$ depends only on the values of the random field within the neighborhood $\mathcal{N}_{\mathbf{x}_i}$ of the site.

The specification of the MRF in terms of local conditional pdfs is cumbersome because the computation of the joint pdf $p(\mathbf{z})$ from the local conditional pdfs is not easy, and the relationship between the local spatial characteristics of a realization and the form of the local conditional pdf is not obvious. MRF can be described by a Gibbs Random Fields (GRF) [71], which overcomes the above problems.

Definition of GRF: The Gibbs distribution, with a neighborhood system $\mathcal{N}$ and the associated set of cliques $\mathcal{C}$, is defined as: $p(z(\mathbf{x}) = \frac{1}{Q}\sum_w e^{-U(\mathbf{z}=\omega)/T}\delta(\mathbf{z}-\omega)$,

where $\delta(\cdot)$ denotes a Dirac delta function, and the normalizing constant Q which makes the sum over all z of $P(z=\omega)$ equal to one, called the partition function, is given by $Q = \sum_w e^{-U(\mathbf{z}=\omega)/T}$ and $U(\mathbf{x})$, the Gibbs potential (Gibbs energy), is defined as $U(\mathbf{z}) = \sum_{C\in\mathcal{C}} V_C(z(\mathbf{x})|\mathbf{x}\in C)$. Each $V_C(z(\mathbf{x})|\mathbf{x}\in C)$, called the clique potential, depends only on the values $z(\mathbf{x})$ for which $\mathbf{x}\in C$. The parameter T, known as the temperature, is used to control the peaking of the distribution. Note that Gibbs distribution is an exponential distribution, which includes Gaussian as a special case.

3.4.7.1 An example of texture segmentation [69]

A random field Y is defined for the textured image that we wish to segment. This random field is a function of the segmentation GRF X and all the texture GRFs $T_1, ..., T_M$ for the M textures in the image. An experiment gives realizations x, $t_1, ..., t_M$ for the random fields X, $T_1, ..., T_M$, respectively. The value of the realization y of Y is defined as follows: if the segmentation x classifies a pixel (i,j) as belonging to texture m, then y_{ij} gets the gray level value of the (i,j) pixel in texture sample m. A maximum a posteriori (MAP) criterion and dynamic programming methods are adapted to determine the segmentation x for a given image y.

Mathematically, the goal is to find x^* that maximizes $P(X = x|Y = y)$. In other words, we want to find the most likely segmentation x^* for a given image y (under the modeling assumptions described above). It should be noted that this segmentation method is currently applicable only to noise-free textured images consisting of known textures and with known high level Gibbs distribution parameters. The parameters are estimated off-line on texture and segmentation samples.

MRF methods can generate a good segmentation in the presence of noise. However, this method needs fine tuning of each parameter, and requires complex computation during optimization of a Gibbs energy function.

3.4.8 Mathematical Morphology

Mathematical morphology is an approach to signal processing in which the manipulation of the data is based on shape [72]. The two basic morphological operators are the erosion and the dilation [33]. By combining these two operations, a whole set of morphological operations can be performed. The application to binary images of morphological operations is simple, but the applications in the field of segmentation remain limited to simple computer vision tasks.

Morphological filtering can be quite effective in simplifying the image characteristics, thus reducing the influence of noise in subsequent processing. The basic morphological tool for image segmentation is watershed. Morphological segmentation relies on on two basic steps: the first, known as marker extraction aims at identifying a set of initial homogeneous zones (whose union does not cover the entire image, but leaves areas of uncertain attribution). The second decision step is based on the watershed process, which attributes the remaining pixels to the appropriate zones. Morphology proves quite efficient in determining the locations of the segmentation, but is weaker in characterizing the properties of each resulting region. Furthermore, the extension of morphological operator to multidimensional features is not trivial.

3.4.9 Neural Networks Approaches

Neural network architectures which help to get the output in real time because of their parallel processing ability, have been used for segmentation and they are robust to noise because of adaptive learning ability [73, 40]. Neural networks are powerful classification tools which encompass many similarities with statistical pattern recognition approaches. They appear to be most powerful when explicit a priori knowledge of underlying probability distributions is unknown. By their "training-by-example" property, neural network classifier may be properly trained to provide outputs that nonparametrically approximate a posteriori class probabilities.

Another important requirement is to have the output in real time. Neural networks are massively connected networks of elementary processors [73]. The massive connectionist architecture usually makes the system robust while the parallel processing enables the system to produce output in real time. The commonly used neural networks for image segmentation are back-propagation neural network and Kohonen's Self-Organizing Feature Maps (SOFM) [74]. A brief review of these networks is presented.

Supervised multilayer feed forward neural networks, also called as back-propagation neural networks, train the networks with error back-propagation manner. There is one input layer, one or more hidden layers and one output layer. Each neuron consists of a set of weights, one for each input, and an activation function. The neuron calculates the weighted sum of the inputs and passes this value to an activation function which generates an output. During training, the network weights are adjusted until some maximum error criterion is satisfied. The training is accomplished with the error back-propagation training algorithm.

$$\delta_{ok} = \frac{1}{2}[(d_k - o_k)(1 - {o_k}^2)], \tag{3.29}$$

$$\delta_{yj} = \frac{1}{2}[1 - {y_j}^2] \sum_{k=1}^{K} \delta_{ok} w_{kj}, \tag{3.30}$$

where w_{kj} is a weight element, d_k is the desired output, o_k is the actual output. The equation for updating the weights for the output layer and hidden layer respectively are:

$$W = W + \eta \delta_o y^t, \quad V = V + \eta \delta_y z^t, \tag{3.31}$$

where η is the learning constant, y is the input to the output layer, z is the input to the hidden layer. After all the input pairs have been presented in the training set, a new set of weights and the outputs are evaluated again. This is repeated until the error reaches a minimum. This is a supervised learning algorithm which performs a gradient descent on a squared error energy surface to arrive at a minimum. For the image segmentation, the number of neurons in the input layer depends on the number of input features for each pixel and the number of neurons in the output layer is equal to the number of classes. The output of the neurons in the output layer has been viewed as a fuzzy set. The weight updating rules have been derived to minimized the fuzziness in the system.

The basic idea of constructing an unsupervised SOFM neural network is to incorporate into the competitive clustering rule some degree of sensitivity with respect to the neighborhood or history using the unsupervised features mapping. The feature mapping can be thought of as nonlinear projection of the input pattern space on the neuron's array that represents features. The projection makes the topological relationship between patterns geometrically explicit in low-order dimensional feature space. The algorithm used to perform feature mapping is rather simple: *winner-take-all.* The details of the SOFM are discussed in Chapter 4.

3.5 Image Sequence Segmentation

3.5.1 Motion-Based Segmentation

Motion based segmentation aims at producing a model that takes in the scene by identifying a certain number of motion components. The modified Hough transform approach [13], the iteration approach [14], and the modified K-means approach [75] are described in the following.

A two stage algorithm that involves a modified Hough transform was proposed in order to reduce the computational burden of the transform [13]. In the first stage of this algorithm, connected sets of flow vectors are grouped together to form components which are consistent with a single parameter set. In order to reduce the computational complexity, several simplications were proposed: (1) decomposition of six parameters into two subset to perform two 3-D Hough transforms, (2) a multiresolution Hough transform, and (3) a mutipass Hough technique, where the flow vectors which are most consistent with the candidate parameters are grouped first. In the second stage, the results of first stage are merged together using several merging criteria. In the final stage, ungrouped flow vectors are assimilated into one of their neighbors.

The iteration method [14] uses a recursive algorithm to detect multiple motions and the corresponding supports between two frames. At each step of the iteration, the frames are analyzed in order to estimate a dominant motion in the scene. The first frame is then warped according to the estimated motion, and the resulting prediction is compared with the original. The areas which are not properly compensated by the dominant motion are labeled, and the estimation step is iterated in order to evaluate a dominant motion for the outliers. The procedure is iterated until the number of pixels or regions marked as unlabeled drops below a threshold.

The modified K-means method [75] produces a description of the video data based on a set of overlapping layers. This description is therefore also referred to as layered description. Each layer consists of three parts: (1) a set of motion descriptors, (2) an intensity map describing the intensity profile of a coherent motion region over many frames, and (3) an alpha plan map describing its relationship with other layers. Each layer corresponds to a set of pixels that move over time according to a model of motion. The multiple motion models are allowed to compete for region support at each iteration.

Motion based segmentation methods rely on only the efficiency of chosen motion models in representing the actual motion present in the scene. In the case of real world sequences with multiple moving objects, quite sophisticated motion models need to be chosen, which increase the computation complexity.

3.5.2 Spatio-Temporal Video Segmentation

Motion is the most significant feature in an image sequences. Therefore, using motion information for segmentation is a good idea that exploits the underlying nature of the video data. The segmentation results rely on only the efficiency of chosen motion model. However, estimated motion fields are not always reliable because of the ill-posed problem of motion estimation. Also, estimated motion vectors around the boundary between two different regions are less accurate because a smoothing (regularization) term is added to circumvent the ill-posed problem of motion estimation [45].

An observation [76] was made that in an image sequence, motion boundaries almost always coincide with intensity boundaries. Therefore, by assuming that motion boundaries coincide with spatial boundaries, some noticeable improvement over motion-only-based methods can be expected if spatial segmentation is used as an

initial segmentation for the motion segmentation. The trend in the field of image sequence segmentation is the support of static segmentation by using static image features such as luminance, texture, so called *spatio-temporal* segmentation method [5, 6, 8, 77, 78].

The spatio-temporal segmentation consists of two step: the first frame is spatially segmented to be initial segmentation and then subsequent frames are merged using affine region matching. Several adjacent regions following a coherent motion may constitute a moving object. From the initial segmented regions, motion information of each region is estimated for motion-based region merging using dense motion at each pixel. In general, affine motion model is used for estimating motion information of each region as follows:

$$\begin{aligned} v_x &= a_{11}x + a_{12}y + b_1, \\ v_y &= a_{21}x + a_{22}y + b_2, \end{aligned} \tag{3.32}$$

where the variables v_x, v_y are the components of the motion vectors, x and y are the coordinates, and $a_{11}, a_{12}, a_{22}, b_1$ and b_2 represent the affine model parameters. The parameters of the motion model can be calculated utilizing a 2-D linear regression. The regression can be evaluated independently, because both equations of the motion model are completely independent.

$$\begin{aligned} a_{11} &= \frac{\sigma({y_i}^2)\cdot\sigma(x_i\cdot v_{xi}) - \sigma(x_i\cdot y_i)\cdot\sigma(y_i\cdot v_{xi})}{\sigma({x_i}^2)\cdot\sigma({y_i}^2) - \sigma(x_i\cdot y_i)^2}, \\ a_{12} &= \frac{\sigma({x_i}^2)\cdot\sigma(y_i\cdot v_{xi}) - \sigma(x_i\cdot y_i)\cdot\sigma(x_i\cdot v_{xi})}{\sigma({x_i}^2)\cdot\sigma({y_i}^2) - \sigma(x_i\cdot y_i)^2}, \\ b_1 &= \langle v_{xi}\rangle - a_{11}\langle x_i\rangle - a_{12}\langle y_i\rangle, \end{aligned} \tag{3.33}$$

where $\sigma(a_i\cdot b_i) = \langle a_i\cdot b_i\rangle - \langle a_i\rangle\langle b_i\rangle$, $\langle x_i\rangle = \frac{1}{K}\sum_{i=1}^{K} x_i$ and K is the number of pixels in a region. The variables a_{21}, a_{22}, and b_2 are calculated in a similar manner. For

every two adjacent regions, the sum of square compensation errors of the two regions when they are compensated using this set of motion parameter is calculated. If the minimum value is smaller than a predefined threshold, the corresponding regions are merged. By this way, we can extract moving objects which have the same motion.

In [79], the algorithm which sequentially refines the segmentation and the motion estimation was proposed by combining static segmentation and motion information in serial manner. In order to efficiently handle camera motion, a global motion estimation is first carried out by a frame matching technique [80] to remove camera motion. After pre-filtering the image using morphological open-close operator, spatial segmentation is performed by K-means clustering algorithm. For each region, affine motion parameters are computed using a matching technique. In the following stage, regions with similar motions are merged by applying a K-medoid clustering algorithm [81] which is less sensitive with respect to outliers. The main advantage of the algorithm is that the boundaries are computed on the luminance signal, and they are precisely located.

In [82], a video object segmentation method using feature fusion and region grouping was proposed. Color is chosen as the major segmentation feature because of its consistency under varying conditions. An iterative clustering algorithm (region merging) is adopted for the segmentation. Optical flow is utilized to project and track the regions. A region grouping is applied at the final stage to avoid oversegmentation and obtain higher level objects. This is realized by using affine motion parameters of each region.

A group processing scheme [21], similar to MPEG-2, is used to refresh the spatial segmentation in order to process the new objects entering the scene and the propagation error due to the affine motion matching. The concept of the method is presented here. Video data are processed in consecutive groups of frames. These groups are

nonoverlapping and independent of each other. The best value of the number of frames in a group depends on object motion activities in the video data. There is one I-frame in each group, which is spatially segmented first. Starting from I-frame, the rest of the frames in the group are segmented consecutively by affine matching the segmented regions to their next frame. These motion segmented frames are called P-frames. Instead of the beginning frame, the middle frame of each group is set as the I-frame. Motion prediction is toward both the forward and backward directions. By this way, the length of motion prediction propagation is reduced by half and segmentation accuracy is improved. The situations of region disappearing and appearing are handled by the adjacent groups provided that the objects stay in the scene for at least a group of frames.

To overcome the problems associated with motion estimation, the moving object tracking algorithm [83] that matches a two-dimensional binary model of the object against subsequent frames using the Hausdorff distance has been proposed. But the algorithm cannot be effective when the image has textured regions since the binary model has been derived from the edge information after thresholding gray-level difference of images.

3.6 Objective Evaluation of Segmentation

While development of segmentation algorithms has attracted significant attention, relatively little effort has been made on their evaluation [84, 85, 86]. For instance, none of the 17 range image segmentation methods listed in [86] has been evaluated using pixel-level ground truth in real images. Also none of the methods has been directly compared to other methods. The situation is almost the same for other applications. None of the papers in the references of this proposal on image sequence segmentations has been evaluated. In [86], the following essential factors were suggested as a guiding philosophy for evaluation of segmentation algorithms.

- *The comparative framework is itself a research issue, and so deserves appropriate conceptual energy in its development.* One surprising thing about many low-level concepts is that there is no rigorous, uniformly accepted definition. For example, the definitions of segmentation are largely similar, but vary in details. Similarly, subjective visual evaluation of results, which has evolved as the norm, should naturally give rise to skepticism. The evaluation procedure should be automated.

- *Metrics are needed for error measurement, in addition to correct/valid performance.* Some types of incorrect/invalid results might be acceptable while others are not. Thus multiple metrics are necessary for potential consumers to make intelligent decisions.

- *The comparative study must use a "large," appropriately designed, real image data set, complete with ground truth.* Evaluations based on one or two images are generally worthless. Work that stops short of using real images inspires little confidence. Establishing ground truth can require some ingenuity and is often painstaking, laborious and time-consuming. However, there simply is no other option.

- *All input data, results and implementations must be made publicly available, both for potential consumers and for true incremental comparisons by others.* Comparison of segmentation results is difficult, since in many cases, we have not been able to reproduce the published results by using authors's algorithm due to the lack of necessary details.

3.6.1 Evaluation Metrics

Evaluation methods can be divided into two categories: analytical methods and empirical methods [85]. The analytical methods directly examine and assess the

segmentation algorithms themselves by analyzing their principles, complexity and properties. The empirical methods indirectly judge the segmentation algorithms by applying them to test images and measuring the quality of segmentation results.

One analytical method has been proposed by taking into account the type and amount of a priori knowledge that has been incorporated into different segmentation algorithms [87]. However, such knowledge is usually heuristic information and different types of a priori knowledge are hardly comparable. Other properties of segmentation algorithms that can be obtained by analysis include the processing strategy, processing complexity and efficiency, and segmentation resolution of algorithm [88]. These properties could be helpful for selecting suitable algorithms in particular applications. For example the processing strategy of segmentation algorithms can be parallel, sequential, iterative or mixed. The parallel algorithms are suitable for fast implementation.

The empirical methods can be divided into empirical goodness methods and empirical discrepancy methods. The empirical goodness methods characterize different segmentation algorithms by simply computing the goodness measures based on the segmented image without the a priori knowledge of the correct segmentation [89]. An adequate segmentation should produce images having higher intra-region uniformity. The uniformity of a feature over a region can be computed on the basis of the variance of that feature evaluated at every pixel belonging to that region [89]. In particular, for a gray-level image $f(x,y)$, let R_i be the ith segmented region, A_i be the area of R_i, then the gray-level uniformity measure (GU) of $f(x,y)$ is:

$$GU = \sum_i \sum_{(x,y)\in R_i} [f(x,y) - \frac{1}{A_i} \sum_{(x,y)\in R_i} f(x,y)]^2. \tag{3.34}$$

It was assumed in [90] that the good segmentation should have smaller number of regions and larger areas with small deviations of features in each region. The

evaluation function F is defined as:

$$F(I) = \sqrt{R} \times \sum_{i=1}^{R} \frac{e_i^2}{\sqrt{A_i}}, \tag{3.35}$$

where I is the image to be segmented, R, the number of regions in the segmentation, A_i, the area of the ith region, and e_i, the feature error of region i (homogeneity measure). The term $\sqrt{R}$ is a global measure which penalizes the segmentation which has too many regions. The term $\frac{e_i^2}{\sqrt{A_i}}$ is a local measure which penalizes small regions or regions with a large deviation of features. The smaller the value of F, the better is the segmentation result.

The disparity between segmentation results and a reference image (ground truth) that is the best expected results can be used to assess the performance of algorithms. Considering image segmentation as a pixel classification process, the percentage of misclassified pixels is the discrepancy measure that comes most readily to mind [91, 86].

Suppose an image consists of N pixel classes, a confusion matrix C of dimension N can be constructed, where each entry c_{ij} represents the number of class j pixels classified as class i by the segmentation algorithms [91]. The segmentation error for each class k is defined as:

$$M^{(k)} = 100 \times [(\sum_{i=1}^{N} c_{ik}) - c_{kk}]/[\sum_{i=1}^{N} c_{ik}], \tag{3.36}$$

where the numerator represents the number of pixels of class k not classified as k and the denominator is the total number of pixels of class k. This is a useful criterion for supervised clustering applications.

Comparison of a machine segmentation (MS) of a range image to the ground truth (GT) [86] is done by representing percentages of overlap with respect to the size of each ground truth region. Five types of region classifications were considered: correct

detection, oversegmentation, undersegmentation, missed and noise. The formulas for deciding classifications are based upon a threshold T, where $0.5 < T \leq 1.0$. Let M and N be the number of regions in the MS and GT, respectively. P_m and P_n be called the number of pixels in each MS and GT region (R_m and R_n), respectively. Let $O_{mn} = R_m \cap R_n$ be the number of pixels whose same image coordinates of both regions R_m and R_n occupy in their respective images.

Correct detection $O_{mn} \geq T \times P_m$ and $O_{mn} \geq T \times P_n$.

Oversegmentation: A region R_n in the GT image and a set of regions in the MS image $R_{m1}, ..., R_{mx}$ where $2 \leq x \leq M$, are classified as an instance of oversegmentation if $\forall i \in x$, $O_{m_i n} \geq T \times P_{m_i}$ and $\sum_{i=1}^{x} O_{m_i n} \geq T \times P_n$.

Undersegmentation: A set of regions in the GT image $R_{n1}, ..., R_{n_x}$ where $2 \leq x \leq M$, and a region R_m in the MT image and are classified as an instance of undersegmentation if $\sum_{i=1}^{x} O_{m n_i} \geq T \times P_m$ and $\forall i \in x$, $O_{m n_i} \geq T \times P_{n_i}$.

Missed classification: A region R_n in the GT image that does not participate in any instance of correct detection, oversegmentation or undersegmentation is classified as missed.

Noise classification: A region R_m in the MT image that does not participate in any instance of correct detection, oversegmentation or undersegmentation is classified as missed.

If for any given region, only one mapping passes its definition, then the classification is done. When two or three mappings pass their definitions for the same region, the the mapping which has the highest average of its metric-pair is taken as the classification. On equal averages, there is bias towards selecting correct detection, then oversegmentation, then undersegmentation.

The comparison works in [86] provided a guiding philosophy for the evaluations of segmentation. However, ground truth for other real images cannot be easily created as that of planar range image segmentation (the evaluations were restricted for only planar range images) and the definition for the over/undersegmentation can be varied in different applications.

The discrepancy measures based only on the number of missegmented pixels do not take into account the spatial information of these pixels. It is thus possible that image segmented differently can have the same discrepancy measure values. To address this problem, discrepancy based on the position of missegmented pixels have been proposed [91]. Let N be the number of missegmented pixels for the whole image and $d(i)$ be a distance metric from the ith missegmented pixel and the nearest pixel that actually is of the miss-classified class. A discrepancy measure, D, based on this distance is defined as:

$$D = \sum_{i=1}^{N} d^2(i). \tag{3.37}$$

This measure is further normalized to eliminate the influence of image size as $ND = 100 \times \sqrt{D/A}$, where A is the total number of pixels in the image.

The above evaluation criteria can create the quantitative and objective ways to judge segmentation algorithms. However, there is no unified evaluation measure which is generally accepted as a good criterion for all the segmentation methods, even if there have been research efforts on the evaluation metrics. It is worth mentioning that to effectively evaluate segmentation algorithms, both their effectiveness in providing information usable by the later goal-oriented interpretation process (evaluation of segmentation at the application stage) [50] and the analysis of segmentation algorithms themselves (complexity, efficiency, segmentation resolution and architecture) should be considered.

Chapter 4

An Image Sequence Segmentation Algorithm Using Multiple Features and Edge Fusion

4.1 Introduction

Semantic object representation is an important step for digital multimedia applications such as object-based coding, content-based access, and manipulations. In this Chapter, a hardware-friendly segmentation algorithm which takes two consecutive frames and extracts bottom level region information by combining static and motion features is proposed [92]. Only reliable motion vectors are employed during the segmentation. Also, the proposed method segments a textured image sequence with low dimensional features. The outputs of the proposed segmentation schemes can be considered as either nonsemantic primitive regions or as region groups. In the initial stage, a multiple feature space is transformed to one dimensional label space by using the self-organizing feature maps (SOFM) neural networks. The next stage is an edge fusion in which edge information is incorporated into the neural network outputs to generate more precisely located boundaries of segmentation. The proposed algorithm differs from existing methods as following: (1) it can segment textured images

with low dimensional features, (2) it leads to more meaningful segmentation region boundaries, and (3) it is easier to be mapped into hardware than existing methods. Experimental results are compared with a segmentation method using evaluation metrics to clarify the advantages of the proposed algorithm objectively.

This Chapter is organized as follows: In Section 4.2 merits and shortcomings of previously proposed segmentation techniques for image sequences are discussed. In Section 4.3 the proposed segmentation algorithm is overviewed. In Section 4.4 the feature selection and filtering, normalization, and weighting methods used in the algorithm are discussed. The feature labeling method using the neural networks are also described in Section 4.4. In Section 4.5 the edge fusion method used in the proposed algorithm which combines a linked edge map and the neural network outputs is described. In Section 4.6 the computational complexity of the proposed algorithm is presented. The segmentation results and their evaluations are presented in Section 4.7. Finally, the concluding remarks are given in Section 4.8.

4.2 Previous Works and Limitations

Hundreds of segmentation methods have been presented in the past, but there is no unified method which can be considered good for all the images, since most algorithms are specific to image formation models, and algorithms developed for one class of images may not always be applied to other classes. The segmentation is still a very challenging topic in the computer vision and image processing community because homogeneous criteria are scene-dependent, no ground-truth segmentation information is available, the number of final segmented regions is usually unknown, and ill-posed problems of the early vision process exist [12].

In order to segment regions for the description of objects in image sequences, we need to choose the homogeneity criteria. They can be a dynamic feature like motion,

static features such as color and texture, or a mixture of both dynamic and static features. Based on the use of features, in general, the image sequence segmentation methods can be grouped into three classes: (1) motion-only-based, (2) two-phase approach using static segmentation followed by motion-based merging, and (3) one-step approach using static and motion features as a feature vector.

If only motion is used for the segmentation, moving objects can be represented directly without bottom level region representation (see Figure 1.1). Many methods have been proposed for segmenting image sequences using only a motion criterion [13, 14, 75, 16, 17]. The methods adapt the principle of K-means cluster algorithm (assuming that a priori K set of model vectors, i.e., K independently moving objects in a scene is known). The segmentation procedure maps the estimated motion vector at each pixel into the label of the model vector which is closest to the estimated motion vector. The segmentation labels can be computed in the least square sense if the estimated motion vectors associated with the respective classes are known.

In general, motion information is very useful for segmentation because motion is one of the principle features in image sequences. These methods can extract moving objects without static segmentation. However, these algorithms suffer from some drawbacks because the estimated motion vectors are not always reliable and motion boundaries do not follow real object boundaries [4]. Since motion alone is employed during segmentation, the results rely only on the chosen motion estimation models. As far as video coding is concerned, the use of unreliable motion does not severely influence the quality of decoded scene as long as the residues of motion compensation are transmitted and the pixel that the vector points to has similar luminance or color. In contrast, the use of unreliable motion vectors creates erroneous segmentation boundaries without sophisticated motion models.

To overcome these drawbacks, two-phase segmentation approaches [78, 21, 93, 19], also referred to as spatiotemporal segmentation, have been proposed assuming

that spatially segmented boundaries coincide with motion boundaries and that the segmented regions have single motion. During the initial phase, spatial gray-level or color segmentation of a frame is first performed, and then the segmented regions are grouped using motion information.

In [78] the segmentation for extracting moving objects consists of three phases: the first frame is partitioned into homogeneous regions based on intensity using a watershed algorithm, four motion parameters are estimated for each region, and motion-based region merging is performed by thresholding motion compensation errors. The spaitial segmentation in [21] employs color and texture features in one scheme. After boundary detection using change in color and texture, disjointed boundaries are connected to form closed contours followed by region merging based on color and texture similarities. Based on affine motion estimation of each region and a group processing scheme similar to MPEG-2, user-defined regions are tracked. A spatiotemporal segmentation technique to identify objects in a video sequence is proposed in [93]. Given oversegmented initial regions, the regions are merged according to their mutual spatiotemporal similarity formulated in statistical framework using a graph-based clustering algorithm. An image segmentation method for separating moving objects from the background is presented in [19]. After extracting a change detection mask using a hypothesis based on variance comparison of intensity differences, foreground regions of spatially segmented regions are tracked over time to extract moving objects. In [8] frame-based spatial segmentation, motion vector based segmentation, and change-detection-based segmentation are combined using a rule-based region processor to extract moving objects. For the spatial and motion segmentation, a recursive shortest spanning tree (RSST) algorithm is used.

Semiautomatic segmentation methods [6, 18] are proposed for semantic video object extraction. An initial object boundary is predefined by user interaction, and

the spatial segmented regions inside the user-defined boundary are tracked. For the spatial segmentations, a morphological watershed algorithm is used in [6]. In [18], nonedge pixels are labeled by using a color quantization map followed by an iterative color merging process. After the color merging process edge pixels are assigned to their neighboring regions.

These two-phase methods can generate more precise boundaries of objects, since a robust spatial segmentation is incorporated as an initial segmentation and the final object boundaries are the union of the spatially segmented regions. However, they have the following drawbacks: (1) unreliable motion vectors may cause to produce erroneously merged regions during the second phase, (2) the presence of textured features in the scene is generally ignored (if not, the dimension of texture feature is high). The homogeniety criterion for the spatial segmentation is luminance or color. Therefore, the results create oversegmentation in the textured area, and there is a need of sophisticated processing to create a more meaningful segmentation in the textured area, (3) they need a large amount of computation, since most of the segmentation algorithms are of a sequential and iterative nature. For real-time applications such as video telephony and live broadcasting, faster segmentation algorithms or specific hardwares need to be developed, and (4) for more meaningful segmented region boundary, edge information should be incorporated during the segmentation process using a more elaborate similarity measure including edge information.

The one-step approach is a multiple feature clustering scheme using static features and motion feature as a feature vector component [94, 7]. This method exploits the merits of the two-phase approach by using the static and dynamic features. However, the differences are this method handles the multiple features in parallel, avoids unreliable motion features during the segmentation, needs a sophisticated feature weighting scheme. This technique can segment textured image sequence with low-dimensional texture features, since motion features are mainly used in the textured

regions. However, in [94] and [7], the iterative process of feature clustering (K-means and fuzzy-C means algorithms) and the need of a predefined number of clusters keep these algorithms from being implemented in a practical system. It is important to evaluate segmentation algorithms objectively in order to provide their effectiveness for next application stages [86]. It should be noted that none of the papers in the above references except [18] (a pixel difference between segmented region and ground truth) has been evaluated.

In this Chapter, a segmentation scheme similar to the one-step approach is proposed. Selected features are integrated by the SOFM neural networks which do not need the iterative process and performs labeling of feature vectors with simple multiplication of feature vectors and trained weights vectors followed by a winner-take-all procedure. In order to generate precisely located object boundaries, edge information is incorporated into the segmentation. The main components of the proposed segmentation algorithm are a feature weighting scheme, neural networks based feature integration, and edge fusion algorithm based on edge linked maps.

4.3 Overview of the Proposed Segmentation Algorithm

The flow chart of the proposed algorithm is shown in Figure 4.1. The selected features consist of pixel-based luminance/chrominance, motion, and two simple texture features. These features are smoothed to remove small details and normalized to remedy different dynamic ranges of the features. A feature weighting scheme based on the motion confidence measure suppresses unreliable feature components. The texture features do not contribute to the segmentation by fixed lower weighting values, and in textured regions, the motion features mainly contribute to the segmentation. The SOFM neural networks [95] transform the multiple feature space into one dimensional label space to generate an oversegmented initial segmentation. An edge fusion

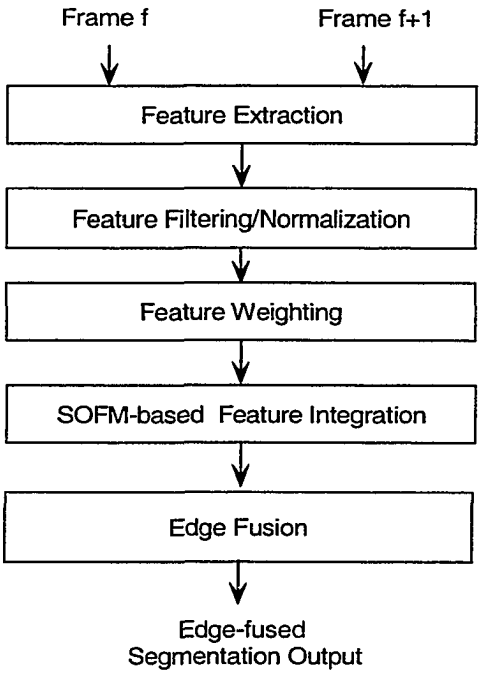


Figure 4.1: Segmentation flow of the proposed algorithm.

process, iterative region merging combining linked edge map, is incorporated into the oversegmented regions in order to produce precisely located object boundaries with lower resolution. Also, the proposed scheme does not need the predefined number of clusters. The main components of the proposed segmentation algorithm are the feature weighting schemes, the neural networks-based feature clustering and labeling, and the edge fusion based on edge linked maps.

4.4 Multiple Feature Integration

4.4.1 Feature Selection

The proposed segmentation scheme relies on seven features extracted from a given color image sequence: three color features (YUV coordinates), two motion features, and two texture features. The feature vector of a pixel (m, n) can be described as $\mathbf{f(m, n)} = [f_y(m, n),\ f_u(m, n),\ f_v(m, n), f_{m1}(m, n),\ f_{m2}(m, n),\ f_{t1}(m, n),\ f_{t2}(m, n)]'$,

where $f_y(m,n)$, $f_u(m,n)$, and $f_v(m,n)$ are luminance (Y) and two chrominance components (UV), respectively, and $f_{m1}(m,n)$ and $f_{m2}(m,n)$ are vertical and horizontal motion, respectively. The $f_{t1}(m,n)$ and $f_{t2}(m,n)$ are the local mean and variance of luminance components defined as follows:

$$f_{t1}(m,n) = \frac{1}{N_w}\sum\sum_{(k,l)\in W} f_y(m-k, n-l), \tag{4.1}$$

$$f_{t2}(m,n) = \frac{1}{N_w}\sum\sum_{(k,l)\in W} (f_y(m-k, n-l) - f_{m1}(m,n))^2, \tag{4.2}$$

where $f_{t1}(m,n)$ and $f_{t2}(m,n)$ represent textural appearance of the area surrounding a pixel (m,n) in a small estimation window centered on this pixel, and N_w is the number of pixel of the window, W. The number of features used for segmentation plays important role on final segmentation results.

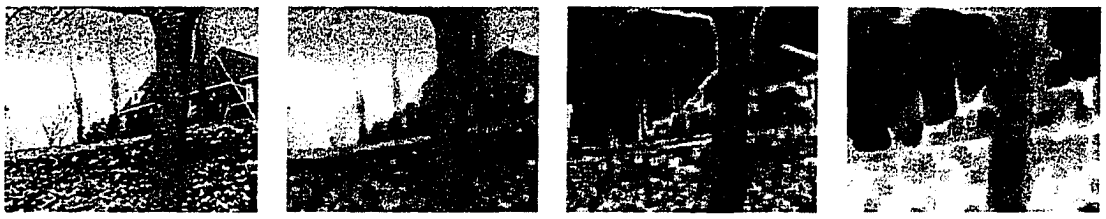
(a) (b) (c) (d) (e)

Figure 4.2: The initial segmentation examples (contours) using different number of feature components. (a) original image, (b) only motion (82 regions), (c) motion + luminance (2634 regions), (d) motion + luminance + texture (2056 regions), (e) motion + luminance + texture + chrominance (1086 regions).

Figure 4.2 shows examples (contour plots) of the SOFM neural network outputs using a different number of feature components without removing small regions. As shown in Figure 4.2 (a), the segmentation using only motion features creates blurred region boundaries due to an ill-posed problem of motion estimation [4]. By combining the static features, the boundaries of segmented regions follow those of the original image. As the feature components are added, the number of regions of segmentations are decreased, resulting in a more semantically correct segmentation.

4.4.2 Feature Filtering and Normalization

In order to remove small details of features in the image which are not relevant to segmentation, median and *min* filtering are applied for extracted features. Luminance/ chrominance and motion feature vectors are smoothed using median filtering, and the two texture features are filtered using *min* filtering. These filtering examples are shown in Figure 4.3. The small details of the tree and flower garden are removed after median filtering. The texture features extracted using local neighborhood operations generate unwanted high values along the region boundaries because a small window straddles them. They are removed by *min* filtering of the extracted texture features as shown in Figure 4.3 (d).



(a) (b) (c) (d)

Figure 4.3: The feature filtering of *flower garden* image. (a) original image, (b) median filtering, (c) texture feature (variance), (d) *min* filtering of the texture feature (the whiter, the higher).

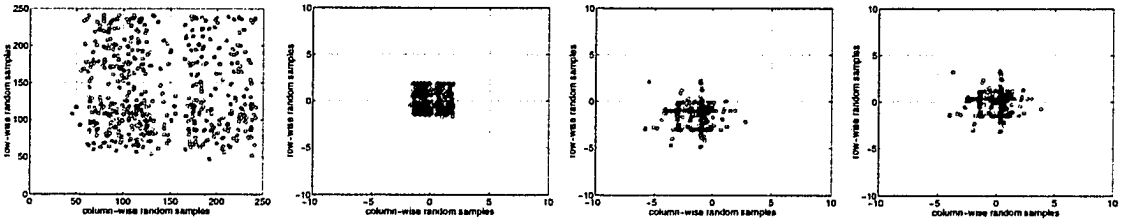
Preparing the feature data for a clustering requires normalization that takes into account the measure of feature distance [63]. For example, Euclidean distance implicitly assigns more weight to features with large ranges than to those with small ranges. Scaling one feature in miles and a second feature in inches makes the second one numerically overpower the first. To remedy this problem, Mahalanobis distance [96] was proposed. Mahalanobis distance scales each feature so that all the features have a unit variance and zero mean. This makes all the features the same importance

while keeping the relative distances of the features. The normalized feature can be expressed as:

$$\breve{f}_i(m,n) = \frac{f_i(m,n) - \overline{f_i}}{\sigma_i}, i = 1, \ldots, N_f, \tag{4.3}$$

where $\breve{f}_i(m,n)$ is a normalized feature i, $f_i(m,n)$ is the filter feature i, $\overline{f_i}$ is a mean of feature i, N_f is the number of feature components, and σ_i is the standard deviation of feature i.

Figure 4.4 shows graphic examples of feature normalization of the *flower garden* image sequence. The dynamic range of the luminance features is from 0 to 255; the dynamic range of the motion features is from -12 to 12. These different dynamic ranges are scaled to be centered around original axis after the normalization as shown in Figure 4.4 (b) and (d).



(a) (b) (c) (d)

Figure 4.4: The feature normalization. (a) luminance features, (b) normalized luminance features, (c) original motion vectors, (d) normalized motion vectors.

4.4.3 Motion Confidence Measures and the Feature Weighting Scheme

Observed motion is not the true motion but apparent motion induced by temporal changes of two dimensional image intensity. Most motion estimation algorithms encounter occlusion and aperture problems because of the ill-posed nature of the motion estimate [4]. Estimated motion vectors around the boundary between two

different regions are less accurate because regularization terms are added for motion estimation to alleviate the nature of the ill-posed problem. In addition, there is greater uncertainty of motion in regions of nontextured features (smooth or constant luminance/chrominance regions) [97].

Some motion estimation algorithms provide a confidence measure for each vector. This measure represents the reliability of an estimated motion vector. Generally, confidence measures are specific to their algorithms, and they lack generality [98]. It is found that for the purpose of image sequence segmentation, the confidence measure of Lucas and Kanade's motion estimation model [46], which is based on a weighted least-square fitting of first order constraint of the optical flow equation, fits well for a region-based block matching technique [47].

The Lucas and Kanade's motion reliability measure is described as follows. A minimum value of two eigenvalues, λ_1 and λ_2 of a matrix Z of the linear system in Equation 4.4 is the reliability measure of the motion estimation.

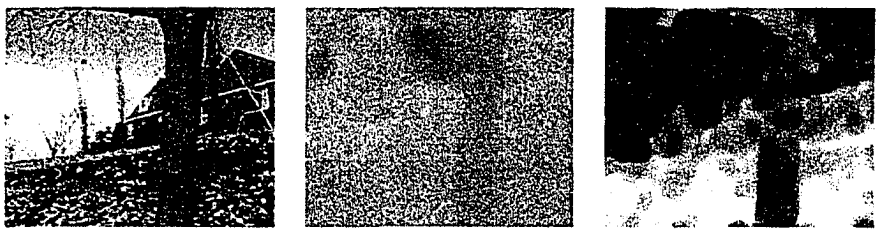
$$Z\mathbf{d} = \mathbf{e}, \tag{4.4}$$

$$Z = \sum_{W} \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix}, \tag{4.5}$$

$$\mathbf{e} = \sum_{W} [I(x,y,t) - I(x,y,t+\tau)] \begin{bmatrix} I_x \\ I_y \end{bmatrix}, \tag{4.6}$$

where I_x and I_y are gradient values of luminance components in x and y directions, respectively, and Z is calculated within a local window W centered on a pixel position, (m,n) where $\mathbf{d}$ is computed, $\mathbf{d}$ is the motion vector. $I(x,y,t)$ is a luminance value in an image at time t. If $mc(m,n) = \min(\lambda_1, \lambda_2) > \lambda$, where λ is a user defined threshold, the motion of the pixel is considered to be reliable. Note that in Equation (4.5), as the summations of both I_x and I_y within the local window are closer to zero (i.e. homogeneous luminance regions, the $mc(m,n)$ is closer to zero), the estimated

motion vector $\mathbf{d}$ is getting more unreliable. On the other hand, the larger $\min(\lambda_1, \lambda_2)$ (larger gradient values in both directions, i.e., high activity of luminance values (textured regions)) gives more reliability to the estimated motions.



(a) (b) (c)

Figure 4.5: The motion confidence measures of *flower garden* frame 1. (a) original image, (b) estimated motion (magnitude after scaled up by 128), (c) *min* filtered motion confidence measures (the whiter, the higher).

Figure 4.5 shows the estimated motion vectors and their confidence measures of *flower garden* frame 1. The sky and the house (nontextured regions) in the image have low confidence values, the tree has higher values, and the garden has the highest confidence values (estimated motions in the garden are homogeneous as shown in Figure 4.5 (b)) since the regions contain many textured features as expected.

From the discussion above, we can devise a weighting scheme which eliminates unreliable motion vectors for the segmentation. Similar concepts of the weighting schemes were used in [94] (hard weighting) and [7] (piece-wise linear weighting). In this Chapter, the more flexible soft weighting scheme based on the motion confidence measures is applied for the segmentation. The feature weighting between luminance/chrominance and motion features is controlled using Equations 4.7 and 4.8.

$$W_m(mc(m,n)) = \frac{1}{1+\exp(\frac{8}{T})}(mc(m,n) - T), \tag{4.7}$$

$$W_{lc}(mc(m,n)) = 1 - W_m(mc(m,n)), \tag{4.8}$$

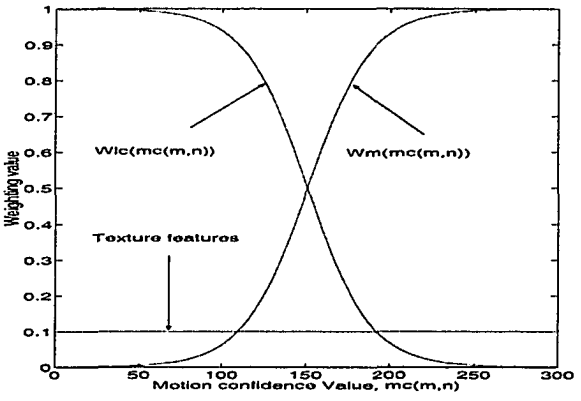


Figure 4.6: The feature weighting scheme.

where $mc(m,n)$, the minimum eigenvalue of the matrix Z, is a motion confidence value of pixel (m,n), $W_m(mc(m,n))$ and $W_{lc}(mc(m,n))$ are the weighting values of motion and luminance/chrominance features, respectively, and T controls the feature weighting.

As shown in Figure 4.6 ($T = 150$), in the textured regions, since the motion confidence measure is high, the motion vectors contribute mainly to the segmentation instead of using the high dimensional texture features, which are usually needed in conventional texture segmentation algorithms. However, small fixed weights (0.1) are applied to the texture features in order to generate smoother segmentation regions, and this effect can be seen in Figures 4.2 (d) and (e). In the nontextured regions (homogeneous luminance/color regions), the luminance/color features are mainly used for the segmentation by suppressing unreliable motion vectors, since the motion confidence measures in the regions are small. Note that the motion confidence measures can be used for the texturedness measure of a scene.

4.4.4 SOFM-based Feature Integration

For segmentation, a multiple feature space is transformed into one dimensional label space. Therefore, we can treat the segmentation task as a clustering of multiple

features. The process of clustering is to find homogeneous clusters of pixels in the feature space and label each cluster as a different region [63]. As iterations go on during clustering, the randomly initialized cluster centers are moved toward the cluster means of feature vectors so that quantization errors, i.e., distance between cluster means and the feature vectors, is decreased. For example, the quantization error of the proposed scheme is decreased from around 12 to 2.27 per pixel after training the neural networks. If we use random cluster centers for segmentation, two different input features which should be labeled differently, can be mapped to the same cluster center.

Conventional clustering algorithms such as K-means and fuzzy C-mean methods are iterative in nature and need a predefined number of clusters. Therefore, it is difficult to implement a practical system using these algorithms. Neural network-based clustering is promising, since its structure is simple and parallel, and segmentation can be done by simple computations between input feature vectors and weight vectors in trained networks. Also, trained network weights can be reusable for other feature vectors for image segmentation. For clustering and labeling the multiple features, unsupervised SOFM neural networks [95] are used. The SOFM neural networks incorporate neighboring clusters for updating cluster means during training. In the previous work of this research, the supervised back propagation neural networks were used for image sequence segmentation [1]. It is a faster solution than an SOFM approach since the labeling process (retrieving neural networks) needs just multiplications without a decision step (one output layer) which is necessary in the SOFM neural networks. However, the labeling of predefined regions by human judgement is prone to errors. This causes the simpler neural networks to generate spurious segmentation results in some images. The segmentation results are attached in Appendix C.

The principle goal of SOFM algorithms is to transform an input feature vector of arbitrary dimension into a one- or two-dimensional discrete map. The most essential feature of an SOFM is to incorporate into the competitive learning rule some degree of sensitivity with respect to the neighborhood. This provides a way to avoid totally unlearned neurons and helps enhance certain topological property that should be preserved in the feature mapping (data clustering) [73][99]. The architecture of SOFM

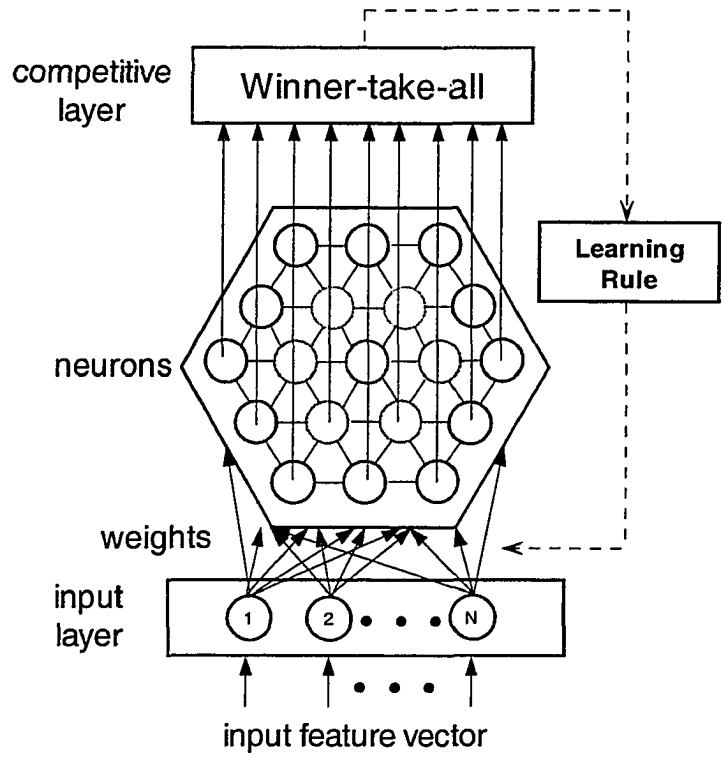


Figure 4.7: The SOFM neural network architecture.

neural networks is shown in Figure 4.7. The most essential feature of the SOFM training is to incorporate neighborhood neurons into the competitive learning rule, which allows a winning neuron to interact laterally. The winning neuron is decided by a minimum Euclidean distance criterion between an input feature vector and weights of the neurons in the networks. Weights of neighborhood neurons around winning

neuron are also updated. The size of the neighborhood slowly decreases with each iteration. The training procedures of the SOFM neural networks are as follow:

Initialization: Choose random values for the initial weight code vectors $\mathbf{c}_j(0)$. The only restriction here is that the $\mathbf{c}_j(0)$ should be different for $j = 1, 2, \ldots, N_c$. N_c is the number of neurons in the lattice (code book size).

Similarity Matching : find the best matching (winning) neuron of an input feature vector, f at time t, using minimum-distance Euclidean criterion:

$$\arg\min_j \|\mathbf{f} - \mathbf{c_j}\|, j = 1, 2, \ldots, N_c. \tag{4.9}$$

Updating: Let i^* denote the index of the winner and let I^* denote a set of indexes corresponding to a defined neighborhood of winner i^*. Then the weights associated with the winner and its neighboring neurons are updated by

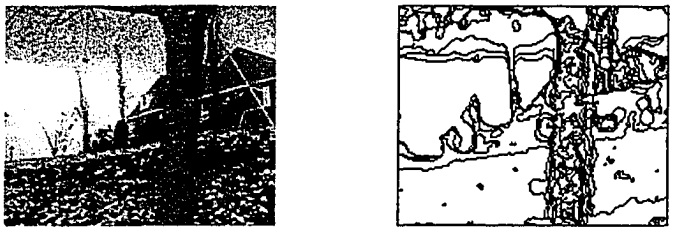
$$\Delta\mathbf{c}_j = \eta(\mathbf{f} - \mathbf{c_j}), \tag{4.10}$$

for all the indexes $j \in I^*$ and η is a small positive learning rate. The amount of updating may be weighted according to a neighborhood function $\Lambda(j, i^*)$

$$\Delta\mathbf{c}_j = \eta\Lambda(j, i^*)(\mathbf{f} - \mathbf{c_j}), \tag{4.11}$$

for all j. The convergence of the feature map depends on a proper choice of η. One plausible choice is that $\eta = \frac{1}{t}$, where t denotes the iteration number. The size of neighborhood should decrease gradually.

Continuation: Continue similarity matching and updating until no noticeable changes in the feature map are observed.



(a) (b)

Figure 4.8: The SOFM outputs of a *flower garden* frame. (a) original image, (b) SOFM outputs (contours).

In the retrieving phase, all the output neurons calculate the Euclidean distance between the trained code weights and the input vector, and the winning neuron is the one with the shortest distance (Equation 4.9). For segmentation, the index of the winning neuron is used as a label.

In the proposed approach, input feature vectors for testing image sequences are taken from sequences which have not been used for training to test the generality capability of the neural network. Figure 4.8 shows an example of an SOFM testing example for a gray-level *flower garden* frame. For the training, the code book size is selected to be 36, and the quantization error of the trained networks for six training images is decreased from around 12 to 2.27 per feature vector after 70,000 iterations. The SOFM outputs follow original region boundaries well.

Note that the segmentation results of the garden regions is not oversegmented. This is because the motion confidence measures in the regions are high so that the motion features mainly contribute to the segmentation. The sky regions are segmented by homogeneous luminance features, since these regions have low motion confidence values as shown in Figure 4.5. However, the tree trunk region is oversegmented because the motion confidence measures are much lower than those of the garden, and the segmentation mainly uses luminance features in this region where their activity

is higher than those of the sky regions. Note that small detailed regions of the image are removed by filtering the feature components before segmentation.

4.5 Edge Fusion

The neural network outputs can generate oversegmented results since several different image sequences are trained which have different cluster means. In order to cover a wide range of cluster means of trained features, the neural networks use a larger code book than the final number of clusters required for each image sequence. In the early stage of this research, a simple region merging process without edge information has been proposed [100, 101, 2]. The segmentation results of this approach are attached in Appendix D. For more meaningful segmentation, the oversegmented result of the neural network outputs are merged in the edge fusion process [102, 103, 104].

The edge and contour information are very much appreciated by the human visual systems and are responsible for our perception. Therefore, the fusion of edge information with the neural network outputs can generate more meaningful segmentation regions. It is desirable to use closed contours to represent objects in the segmentation process. The majority of edges produced by an edge detection algorithm is often nonclosed. An edge linking algorithm proposed in [39] is used to generate closed contours before the edge fusion process begins. The initial edge map was generated by a method proposed in [38].

Two criteria, connectivity measures [50] and boundary strength [105], are adapted to the edge fusion process. The connectivity measure criterion is based on the heuristic that a pair of regions sharing a relatively large common boundary are good candidates for merging. The smaller this criterion, the less likely it is that two regions should

be merged. This measure discourages the formation of regions with small necks connecting larger areas. This criterion is further modified to be bounded between 0 and 1 as shown in the following equation:

$$C_{ij} = \frac{CB_{ij}}{\min(P_i, P_j)}, \tag{4.12}$$

where CB_{ij} is the length of the common boundary separating two regions i and j. P_i and P_j are the perimeters of regions i and j, respectively. Figure 4.9 shows examples of a weak and a strong geometric connectivity case. The two regions with weak geometric connectivity are unlikely to be merged. On the other hand, those with strong one are likely to be merged. Boundary strength criterion measures the

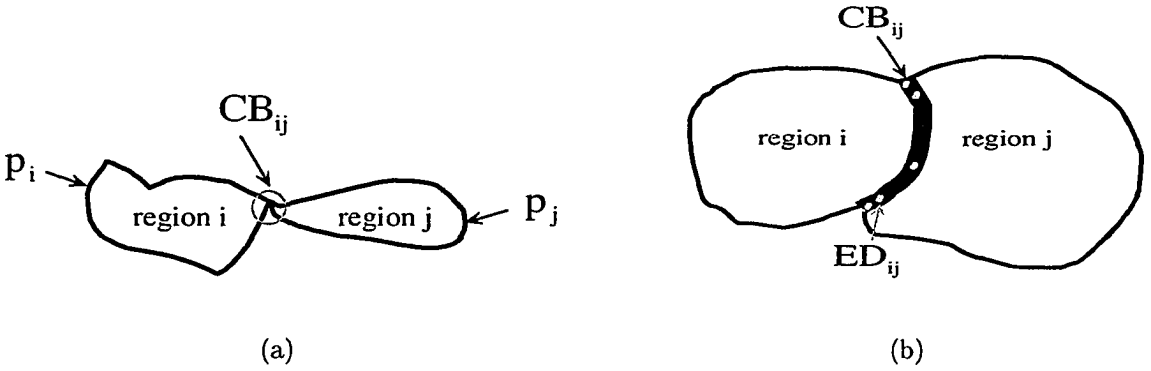


Figure 4.9: The geometric connectivity and boundary strength. (a) weak geometric connectivity, unlikely to be merged, (b) strong geometric connectivity and weak boundary length, likely to be merged.

strength of the boundary between two regions. The larger the number of edge points on the boundary, the larger the boundary strength is. The boundary strength is defined as follows:

$$S_{ij} = \frac{ED_{ij}}{CB_{ij}}, \tag{4.13}$$

where ED_{ij} is the number of edge points on the common boundary between two regions. The boundary strength S_{ij} is bounded between 0 and 1. Figure 4.9 (b)

shows an example of weak boundary strength where there are not many edge points on the common boundary between regions i and j. The two regions with weak boundary strength are likely to be merged. On the other hand, two regions with strong boundary strength are not likely to be merged.

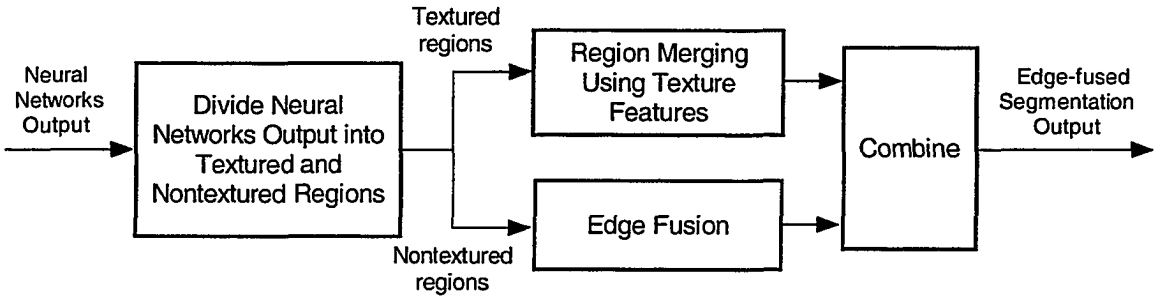


Figure 4.10: The flow diagram of edge fusion.

The flow diagram of the proposed edge fusion algorithm is shown in Figure 4.10. The edge fusion process starts by dividing the neural network outputs into textured and nontextured regions in order to apply edge fusion for nontextured regions, since the edges of textured regions do not provide meaningful region information. For the division, the texturedness measures, i.e., motion confidence values, are thresholded by a predefined value chosen by the user.

The nontextured regions are further split by the linked edge map by overlapping the contours of the neural network outputs and the linked edge map followed by a connect component analysis of the overlapped binary image. A set of attributes is calculated for each region, and a merging process begins to produce the final segmentation map. After splitting regions of neural network outputs using the edge linked map, the following edge fusion algorithm is applied:

`divide neural network outputs into textured and nontextured regions.`

`split nontextured regions using edge linked map.`

`initialize a merged region counter,` N_{merg}.

```
while(N_merg > 0)
 calculate the region attributes; sort region sizes.
 for i=smallest region to largest,
  find neighbor regions (j = 1,...,N_n) of i.
  exclude neighbor regions with C_ij < 0.1; compute the S_ij.
  calculate the energy, E_ij = FD_ij + α · sign(S_ij − β).
  find the neighbor region ĵ of a minimum negative energy value.
  ĵ = arg min_j(E_ij ≤ 0).
  if ĵ exists or region size of i is less than TH_rs,
   merge region i and ĵ; N_merg = N_merg + 1.
  end
 end
end
```

N_n is the number of neighboring regions of region label i. The region attributes consist of the means of YUV features and size. FD_{ij} is the difference of mean values of luminance/chrominance in region i and j, where region j is a neighbor of region i. α, a segmentation resolution parameter, controls the energy values, and the parameter β reflects the degree of the boundary strength in the energy equation by which the current region and a neighboring region are merged. For example, if β is less than 0, the term $\alpha \cdot sign(S_{ij} - \beta)$ never goes to the negative value, and the corresponding energy values always stay positive. Therefore, no candidate neighboring regions will be merged. Neighboring regions with weak geometric connectivity (less than 0.1) are excluded from the computation of the energy values. Also, small regions (less than a predefined threshold TH_{rs}) are merged into a neighboring region $\hat{j}$. Note that only when S_{ij} is less than β and FD_{ij} is less than α and the energy E_{ij} has a negative

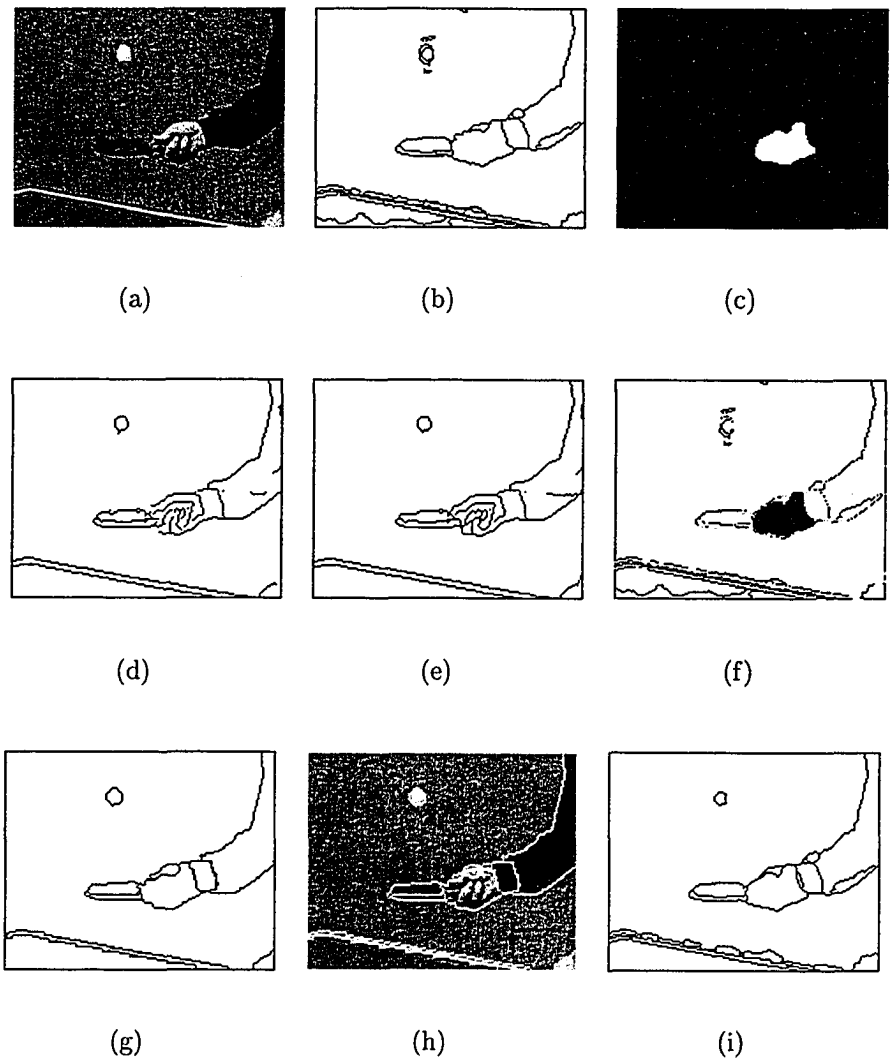


Figure 4.11: An example of edge fusion process. (a) original image, (b) SOFM outputs, (c) the division of nontextured and textured regions (black: nontextured regions, white: textured regions) (d) edge map, (e) linked edge map, (f) overlapping the SOFM outputs and linked edge map, (g) edge fusion result ($\beta = 0.4$), (h) overlapping the edge fusion result onto the original image (i) nonedge fusion result.

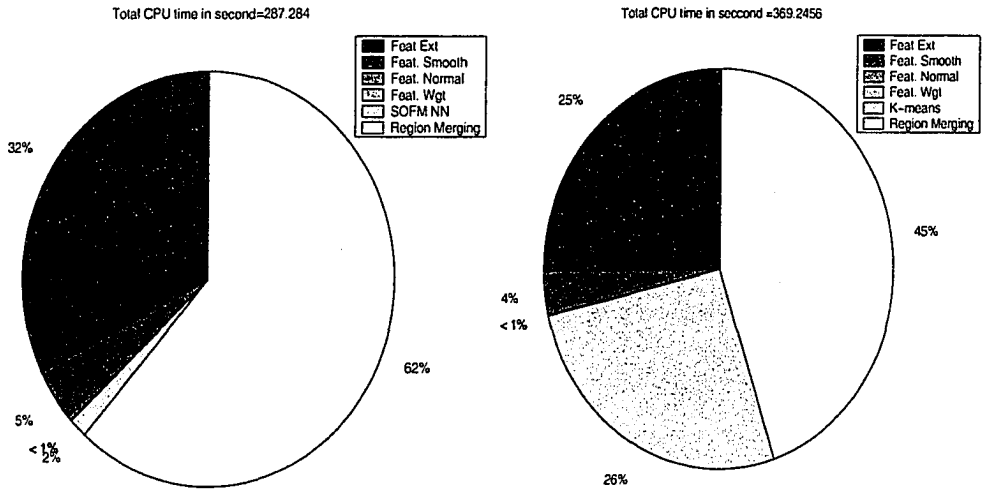
value, are two regions, i *and* $\hat{j}$, to be merged. This eliminates the use of spurious edges between the regions which have smaller luminance/chrominance difference.

Figure 4.11 shows an example of edge fusion process for *table tennis* sequence 1. The neural network outputs are further divided into nontextured and textured regions as shown 4.11 (c) (the hand regions have textured features). The linked edge map has more closed contours as shown in Figure 4.11 (e). We can clearly see the difference of region contours between the neural network outputs and linked edge map (gray values). The final result of edge fusion output is compared with a result using a region merging technique without edge information [2] in Figures 4.11 (g), (h), and (i), respectively. The edge fusion process generates more meaningful segmentation whose boundaries are closer to the original edge map than those of the nonedge fusion segmentation.

4.6 Computational Complexity of the Segmentation Algorithm

The proposed segmentation algorithms using both nonedge [2] and edge fusion process [92] were implemented in MATLAB. The programs ran on a 500 MHz Pentium III PC with 192 MB memory. In this Section, the average CPU times of both algorithms for segmenting a frame are collected in order to observe and compare running times of all subtasks. Also, the computational complexity of the proposed neural networks-based clustering and labeling scheme are compared with a typical K-means algorithm.

Figure 4.12 (a) shows the average CPU time broken down into average CPU time for subtasks in case of nonedge fusion process. The feature extraction, feature smoothing, the SOFM neural network clustering and the region merging process take 32%, 5%, 2%, and 62% of the CPU time, respectively. The CPU time for the feature



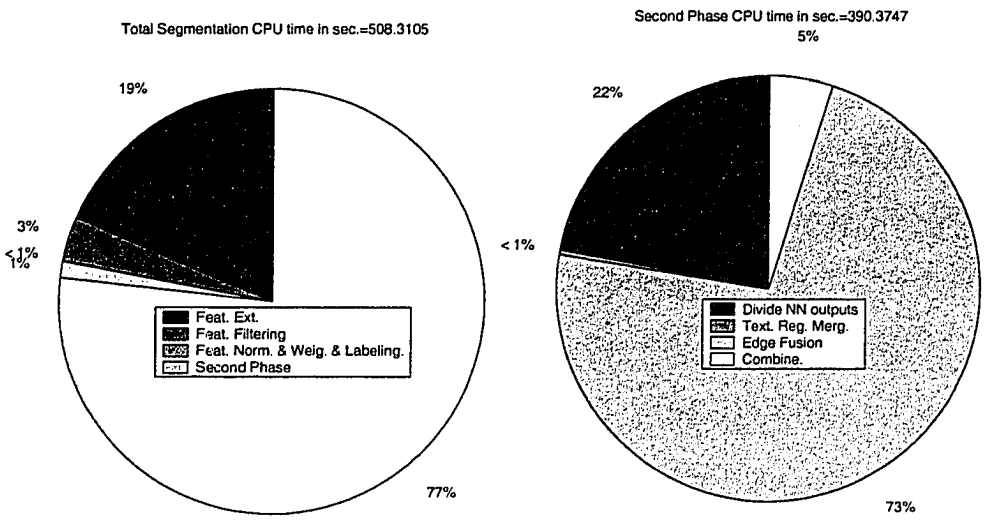
(a) proposed. (b) K-means algorithm.

Figure 4.12: Comparison with the CPU times of the proposed (nonedge fusion) and K-means algorithms.

normalization and weighting procedure are negligible. 91% of the CPU time for feature extraction was spent on motion estimation, which uses a three level hierarchical matching method. The region merging process for the gray images consumes 62% of the CPU time (176.79 sec.). This is because the merging process is iterative in nature. The SOFM subsystem takes a very small portion of the CPU time (2%, 4.50 sec.) because the feature quantization process is simple (matrix multiplications followed by a minimum operation for the winner-take-all). The proposed segmentation algorithm in CPU time is compared with an implementation of the K-means algorithm as the clustering method where the iterations of K-means algorithm were stopped whenever the quantization error of each pixel is the same as that of the SOFM neural networks during iterations, and the codebook size is the same as the SOFM networks. All the other procedures are the same as experiments of the proposed method. Figure 4.12 (b) shows the average CPU time broken down into average CPU time for subtasks. The total CPU time for this system is increased to 369.25 second. The CPU time for

the K-means algorithm itself is 97.49 seconds (26%) on average, which is much slower than the SOFM neural networks implementation. The merging process takes 165.76 seconds (45%) which is little less than that of the SOFM networks.

Let the number of pixels in an image be N, and let the maximum possible number of regions labeled in the SOFM neural network outputs be M. Also, let the codebook size and the number of feature components be N_c and N_f, respectively. The computational complexity of the SOFM neural networks is $O(NN_cN_f + {N_c}^2)$, where NN_cN_f is for the matrix multiplications between input feature vectors and the codebook for whole image, and ${N_c}^2$ is for the winner-take all procedure of the multiplication results. The winner-take all procedure needs the sorting of the N_c multiplication results (bubble sorting). The computational complexity for the merging process is $O(M^2)$ per iteration. This is because both sorting of region size and feature distance calculations need $O(M^2)$ operations.



(a) entire system. (b) second phase.

Figure 4.13: CPU time analysis of the proposed algorithm using edge fusion.

The average CPU time for segmentation of a QCIF color frame was evaluated using pure software implementation in case of edge fusion process. Figure 4.13 shows

the amount of CPU time for each subtask. The second phase of the segmentation process (edge fusion) consumes 77% of the CPU time.

The feature labeling subsystem takes a very small portion of CPU time (less than 1%) because the feature labeling process of the SOFM neural networks is simple and parallel. The use of the neural networks for quantization and labeling of the multiple features takes advantage over conventional iterative quantization algorithms which may take a larger portion of the CPU time. As shown in Figure 4.13 (b), 95% of the CPU time for the second phase of the segmentation is taken by the division of the neural network output into textured and nontextured regions and iterative edge fusion process. The textured region merging takes less than 1% of the CPU time in the second phase. It is clear that in order to achieve real-time image segmentation, a special-purpose VLSI architecture must be developed, especially for the edge fusion process.

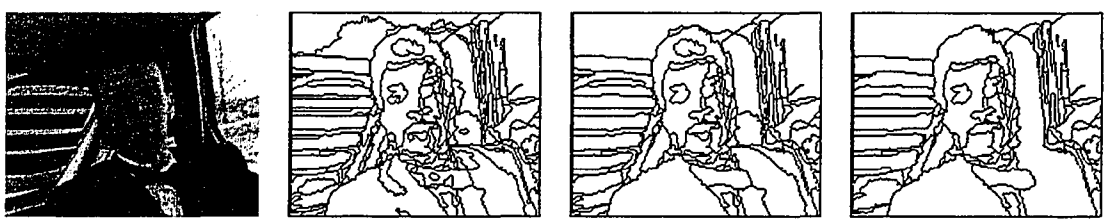
4.7 Experimental Results and Evaluations

The first two frames of five gray-level image sequences, *calendar and train, Claire, football, mall, and MIT*, are used for the SOFM training; and another five image sequences, *flower garden, Miss America, salesman, Susie, and table tennis*, are used for testing the segmentation of gray-level image sequences. In addition, six color image sequences, *mobile, Claire, mall, football, MIT, and flower garden*, are used for the SOFM training, and another six color image sequences, *car phone, foreman, Miss America, mother&daughter, salesman, and table tennis*, are selected for testing the segmentation of color image sequences.

Dense motion vectors are estimated by using Anandan's region matching motion estimation methods [47]. Neighborhood operations using 5×5 local windows are used in order to extract the features and to smooth them. For the edge fusion, high

threshold of 15 and low threshold of 10 are selected to generate edge linked map [38, 39]. For the feature weighting, T is selected to be 150. The threshold value for C_{ij} is set to be 0.1 and β is set to be 0.4.

The code book sizes of the SOFM neural networks for gray-level and color image sequences are selected to be 6×6 and 7×7, respectively. The number of epochs for the SOFM neural network training is 70,000. The training is divided into two phases, the ordering phase and the tuning phase. During the ordering phase ($1 \sim 1,000$ epochs), the learning rate is decreased from 0.9 to 0.02, and the neighborhood distance is adjusted from maximum neuron distance to 1. During the tuning phase, the learning rate continues to decrease from 0.02 very slowly. The neighborhood function is 1 for a winning neuron and 0.5 for neighborhood neurons. After training the neural networks, the same code vectors are used for the segmentation of all testing image sequences.



(a) (b) (c) (d)

Figure 4.14: The segmentation resolutions with different α values. (a) original car-phone image (frame 1), (b) $\alpha = 7$, (c) $\alpha = 13$ and (d) $\alpha = 17$.

4.7.1 Effect on the α Values

Parameter α used for the edge fusion controls segmentation resolution. The Figure 4.14 shows segmentation results using different α values. When α is increased, the result of segmentation has lower resolution, thus, has a smaller number of regions.

4.7.2 Segmentation Results of Different Frames in an Image Sequence

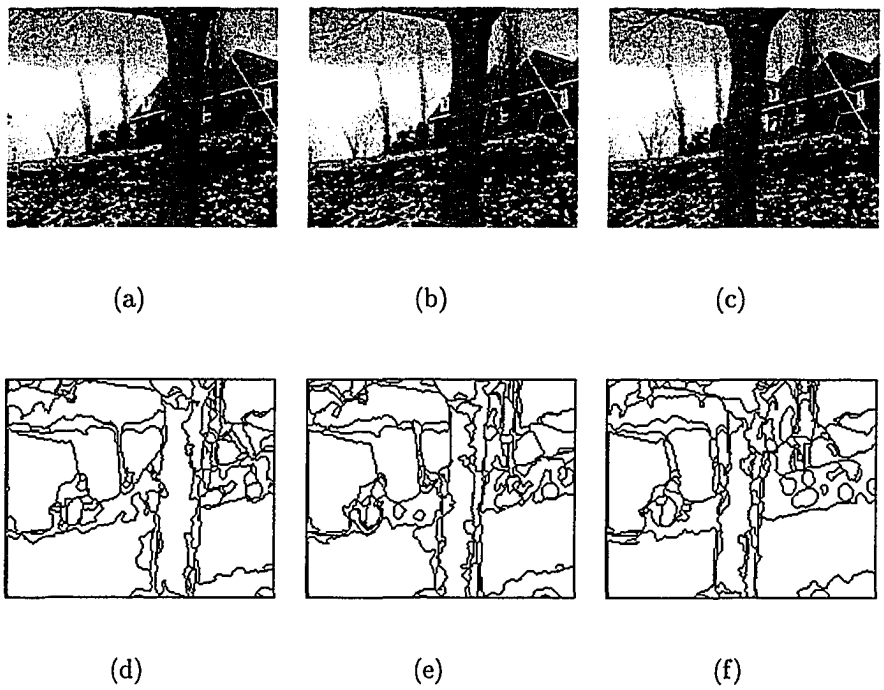


Figure 4.15: The segmentation results of different frames in the *flower garden* image sequences. (a), (b) and (c) original frames 1, 5, and 10, respectively. (d), (e) and (f), segmentation results of frame 1, 5, and 10, respectively.

Since frames of a scene have strong spatial and temporal correlation in general, we do not need to segment every frame of the same scene for some applications [21]. For segmentation of the next frame, motion projection of current segmented regions can be used. For the applications of semantic object extraction, it is desirable to segment every frame for more meaningful representation, since the continuous motion projections of segmented regions of a frame does not always guarantee meaningful region boundaries of the next frames. By combining the motion projection and spatial segmentation of every frame [18], objects in a scene can be better represented. Figure 4.15 shows the segmentation examples of different frames for the *flower garden* sequence. The moving tree is segmented well in frames 1, 5, and 10.

4.7.3 Segmentation Results of a Variety of MPEG Image Sequences

The segmentation results for both gray-level and color image sequences of the first frame are shown in Figures 4.16 and 4.17. Also, for the sake of comparison, an image segmentation method proposed by Dorin and Peter [106] is chosen. Undersegmentation parameters are selected for Dorin's method, since two other classes (oversegmentation and quantization) generate oversegmentation results. These results are shown in Figures 4.16 and 4.17 along with the segmentation results of the proposed method. It should be noted that the gray-level image sequences and the color image sequences used for these experiments were acquired at different sampling frequencies. Therefore, the texturedness in the same image sequence can be different in the gray-level and color image sequences, even though they are the same scene (compare the gray level and color table tennis image sequence).

In the *flower garden* sequence as shown in Figure 4.16, the tree moves with almost pure translation motion over the background. The tree, sky, and house have nontextued regions, and the garden has textured features. Therefore, luminance features in the tree, the sky, and house regions, and motion features in the garden are mainly contributed to the segmentation. Note that the garden is segmented well with several regions instead of many small regions. Use of the gray-level feature only in Dorin's method generates oversegmentation results in the background as shown in Figure 4.16, the boundaries of the tree are not precisely located, and the tree is merged with the house. On the other hand, the proposed scheme generates more reasonable results because the tree and background are generally well extracted. The *Miss America*, *salesman*, *surfside*, and *Susie* in Figure 4.16 do not have many textured regions. Therefore, luminance features are mainly contribute to the segmentations. The *table tennis* sequence in Figure 4.16 has a very fine textured background in the

Figure 4.16: Gray-level image segmentation results. First column: original images (frame 1), second column: proposed method, third column: Dorin's method. From the first row, the results are shown for *flower garden, Miss America, salesman, Susie, and table tennis* image sequence.

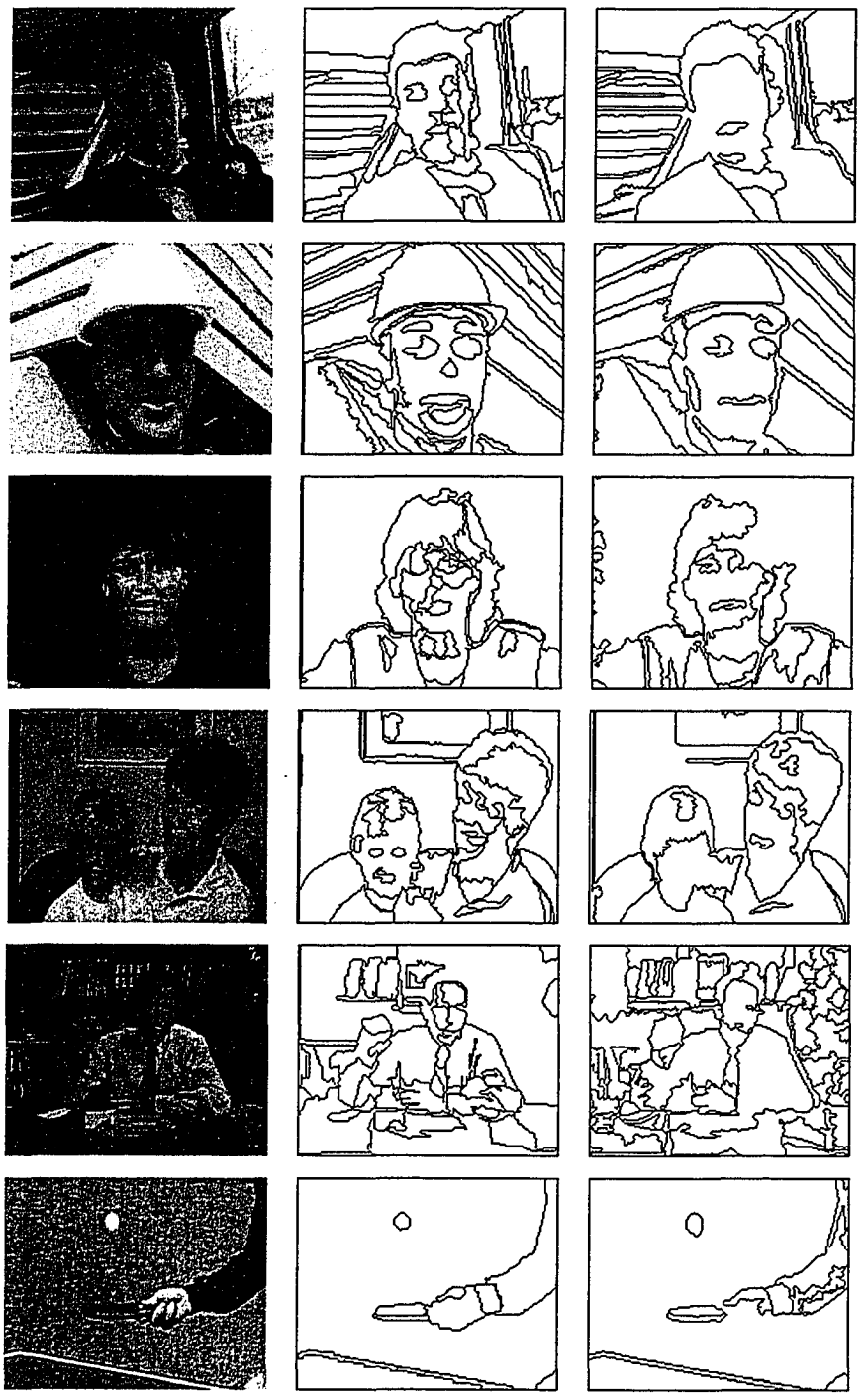


Figure 4.17: Color image segmentation results. First column: original images (frame 1), second column: proposed method, third column: Dorin's method. From the first row, the results are shown for *carphone, foreman, Miss America, mother&daughter, salesman, and table tennis* image sequences.

original images. Therefore, most of the regions are segmented using reliable motion features which is almost zero (background does not move in this sequence), and the segmentation has one background region instead of many small regions in the Dorin's segmentation.

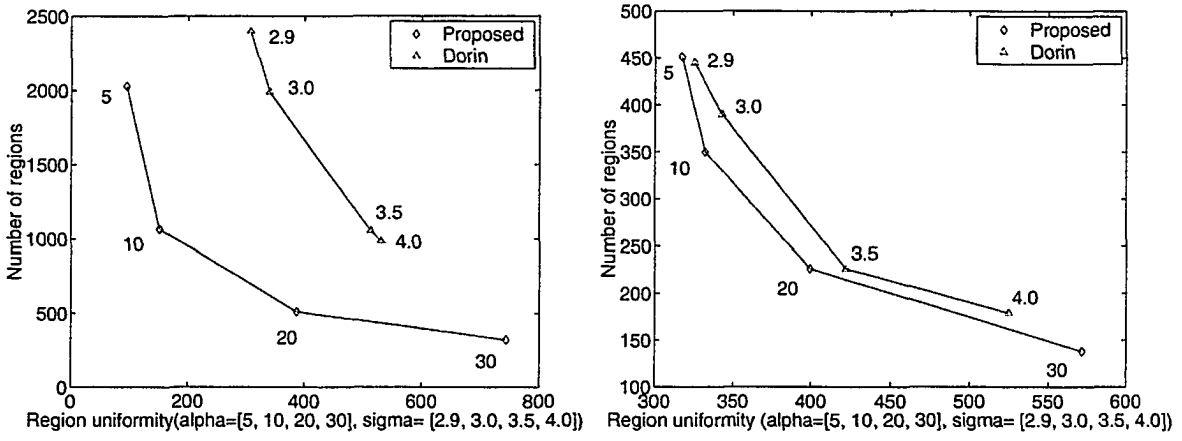
The color image sequences in Figure 4.17 do not have many textured regions. Therefore, the proposed scheme uses mainly static color features for these segmentations. It is worth mentioning that the hand region in the table tennis sequence is segmented as one region instead of several smaller regions, since the region has a textured feature, and the motion feature mainly contributes segmentation of the region. As shown in Figure 4.17, the hand region of the undersegmented result of the table tennis sequence using Dorin's method is merged with the background. However, the proposed scheme extracts the boundaries well, since reliable motion features are used for the segmentation of the region (regarded as a textured region).

For the objective evaluations of the segmentations, evaluation metrics using goodness measures and ground truth information are applied.

4.7.4 Evaluation of the Segmentation Results

For the goodness measures without using ground truth, the gray-level uniformity (GU), and color uniformity (CU) (the summation of luminance/chrominance variance of each region) [89], and $F(I)$ [90] metric are modified. The GU and CU values are normalized by total pixels of an image, and for the computation of $F(I)$, e_i^2 is replaced with the mean variance of the luminance/color in nontextured regions and that of motion in textured regions, respectively. Also, the $F(I)$ value is scaled by 100. The smaller values of these goodness measures are considered to be better segmentation.

The segmentation results of both methods using the same image sequences as in Figures 4.16 and 4.17 are evaluated using a gray-level/color uniformity vs. the number



(a) gray-level image sequences.

(b) color image sequences.

Figure 4.18: Gray-level/color uniformity vs. number of segmented region plots.

of segmented regions plot using the same small region threshold. The segmentation resolution parameters were varied from low to high in order to determine the effect of the parameters.

The range of the parameters are $\alpha = [5,\ 10,\ 20,\ 30]$, and $\sigma = [2.9,\ 3.0,\ 3.5,\ 4.0]$ for the proposed algorithm and Dorin's method [106], respectively. The uniformity of the segmented regions is expressed using the summation of variance of the all the segmented regions divided by total number of pixels. In order to calculate the uniformity, we use the mean variance of the luminance/color features in nontextured regions and that of motion features in textured regions, respectively. A better segmentation method should have the product of the number of regions and the uniformity closer to the origin (0,0) of the variance-number of regions plane. As the segmentation resolution is decreased, the number of segmented regions is decreased and the variance is increased.

Figure 4.18 shows that the results from the proposed segmentation algorithm are better for both gray-level and color image sequences, especially much better for the gray-level image sequences which contain more textured features in the original

images than the color image sequences. The detailed data for gray-level and color segmentation are shown in Tables 4.1 and 4.2, respectively. We use the product of the number of segmented regions and the uniformity as an overall figure of merit (FOM). The proposed segmentation results are 68.2% and 8.4% better in terms of FOM than those of Dorin's for four segmentation resolutions of the gray-level and color image sequences, respectively. For the medium segmentation resolution ranges ($\alpha = 10,\ 20$, and $\sigma = 3.0,\ 3.5$), which can be considered as typical for object-based applications, the proposed algorithm is 70.5% and 10% better than that of Dorin's algorithm for the gray-level and color image sequences, respectively. The detailed evaluation data are attached in Appendix E.

Table 4.1: The number of segmented regions and gray-level uniformity data

Proposed algorithm				Dorin and Peter's algorithm			
α	Regions	Uniformity	FOM	σ	Regions	Uniformity	FOM
5	2030	93.76	1.91e5	2.9	2397	306.39	7.35e5
10	1065	151.65	1.62e5	3.0	1990	338.57	6.75e5
20	510	386.12	1.97e5	3.5	1053	513.02	5.41e5
30	320	743.90	2.39e5	4.0	985	530.49	5.23e5
total	3925	1375.42	7.87e5	total	6425	1688.45	24.71e5

Table 4.2: The number of segmented regions and color uniformity data

Proposed algorithm				Dorin and Peter's algorithm			
α	Regions	Uniformity	FOM	σ	Regions	Uniformity	FOM
5	451	317.20	1.44e5	2.9	445	325.14	1.45e5
10	350	331.19	1.17e5	3.0	390	342.44	1.34e5
20	226	399.92	0.91e5	3.5	226	421.88	0.96e5
30	138	571.67	0.78e5	4.0	179	524.47	0.94e5
total	1165	1619.98	4.29e5	total	1240	1613.91	4.68e5

Also, the segmentation results in Figure 4.16 and 4.17 are compared with Dorin's method for each image sequence using the two goodness measures as shown in Tables 4.3 and 4.4. Most of the proposed segmentation results have better goodness

Table 4.3: The goodness measures of the gray-level image segmentation.

	Flower garden	Miss Am.	Sales-man	Susie	Table tennis	Total	Mean
Regions (proposed)	23	53	74	53	27	230	46
Regions (Dorin's)	210	69	383	137	186	985	197
GU (proposed)	0.292	0.091	0.133	0.081	0.310	0.907	0.181
GU (Dorin's)	1.356	0.115	0.599	0.087	0.287	2.444	0.489
F(I)(Proposed)	26	24	33	21	31	135	27
F(I)(Dorin's)	2301	82	1281	74	482	4420	844

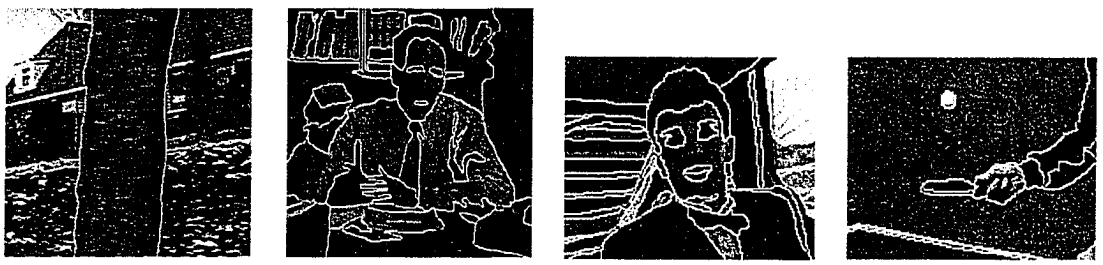
Table 4.4: The goodness measures of the color image segmentation.

	Car-phone	Fore-man	Miss Am.	Mom& dau.	Sales-man	Table tennis	Total	Mean
Regions (proposed)	53	57	39	57	54	18	278	46
Regions (Dorin's)	103	63	71	46	125	12	420	70
CU (Proposed)	0.264	0.183	0.129	0.099	0.195	0.204	1.074	0.179
CU (Dorin's)	0.402	0.237	0.201	0.060	0.286	0.128	1.314	0.219
F(I)(Proposed)	31	25	15	19	26	21	137	23
F(I)(Dorin's)	180	92	219	17	130	15	654	109

measures, especially when the image sequences have textured regions such as the gray-level flower garden and table tennis sequence.

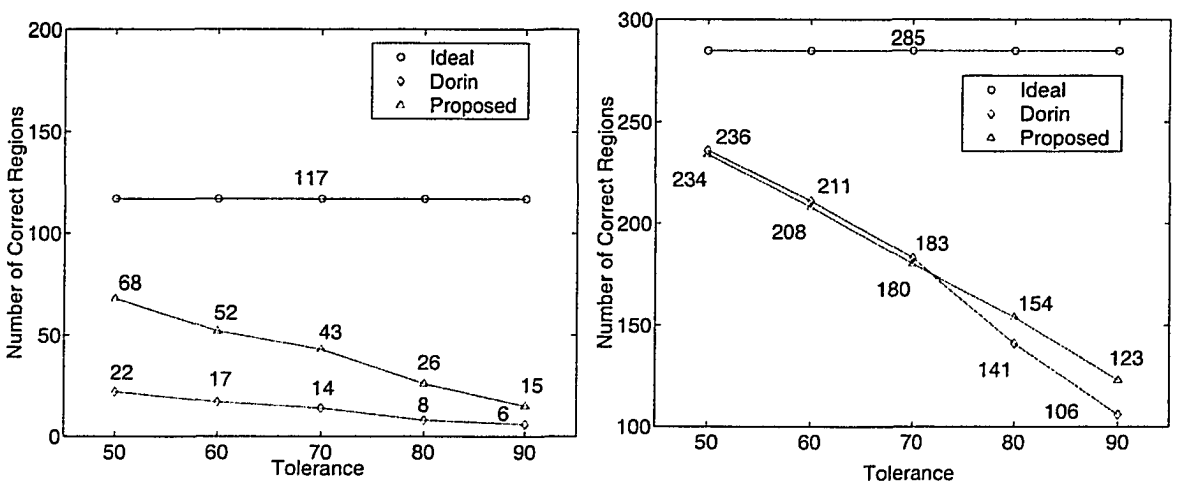
Comparison of machine segmentation (MS) results with ground truth (GT) information can be done by representing percentages of overlap with respect to the size of each ground truth region [86]. The formula for deciding correct segmentation is based upon a tolerance T, where $0.5 < T \leq 1.0$. Let P_m and P_n be called the number of pixels in each MS and GT region (R_m and R_n), respectively. Let $O_{mn} = R_m \cap R_n$ be the number of pixels whose same image coordinates of both regions R_m and R_n occupy in their respective images. The segmentation results can be considered as correct segmentation if the following equation is satisfied.

$$O_{mn} \geq T \times P_m \text{ and } O_{mn} \geq T \times P_n. \tag{4.14}$$



(a) (b) (c) (d)

Figure 4.19: Examples of ground truth data. (a) and (b) gray-level *flower garden* and *salesman* frames, respectively. (c) and (d) color *carphone* and *table tennis* frames, respectively.



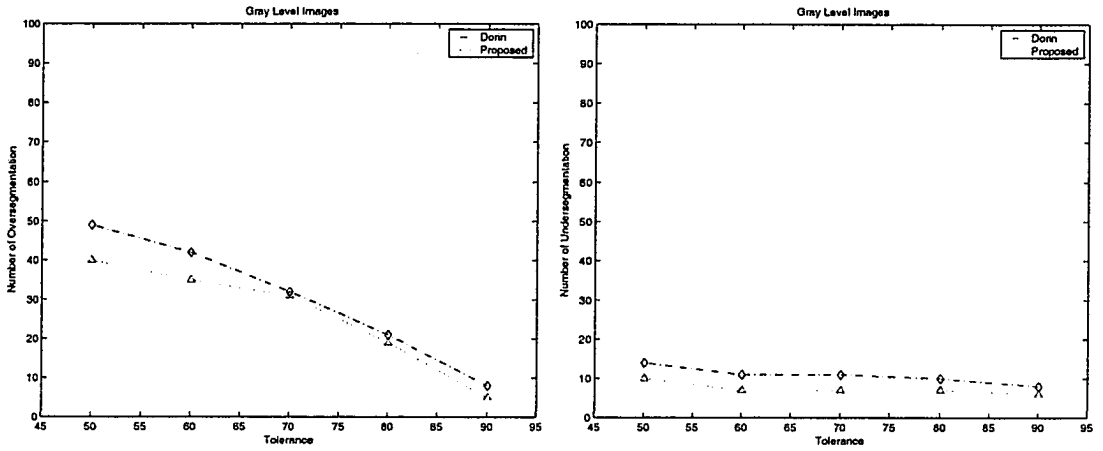
(a) gray-level image sequences. (b) color image sequences.

Figure 4.20: Evaluation of correct segmentation using ground truth.

In addition, the two different segmentation methods of gray-level and color image sequences are evaluated using ground truth data which were generated by hand segmentation as shown in Figure 4.19 (four images out of eleven GT data are shown here) and tested and verified by several human judges. As the tolerance value is increased, correct detection rates are decreased as shown in Figure 4.20. For gray-level images, more regions are detected as correct segmentation in the proposed method. For the color images, the number of correctly segmented regions of Dorin's method is slightly better in case the tolerance is low. For the gray-level images, the overall correct segmentation percentages are 34.88% and 11.46% with respect to the ideal ground truth data for the proposed algorithm and Dorin's method, respectively.

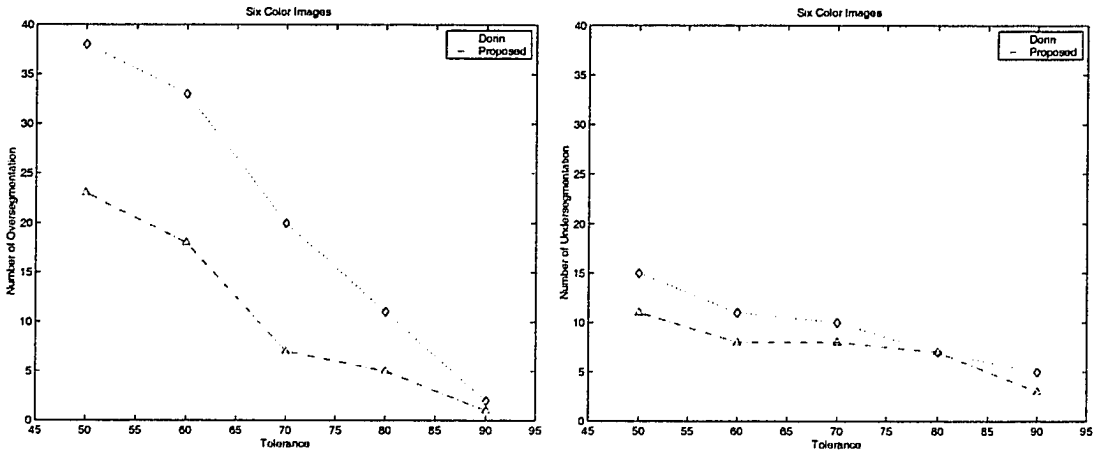
The correct segmentation percentages of the color images are 63.65% and 60.98% with respect to the ideal ground truth data for the proposed algorithm and Dorin's method, respectively. Therefore, the proposed segmentation algorithm has 23.42% and 2.67% increased correct segmentation rates through all the tolerance values than those of Dorin's method for the gray-level and color image sequences, respectively. The detailed evaluation data are attached in Appendix F.

Evaluations for both oversegmentation and undersegmentation results are also performed as shown in Figures 4.21 and 4.22 for gray-level and color image sequences, respectively. The total number of regions in the ideal case for five gray images, 117, is calculated based on the GT regions, not the MS regions (proposed: 268, Dorin: 971) of gray images. Therefore, 151 (proposed) and 854 (Dorin) regions in the MS regions do not participate in this evaluation (undetermined) respectively. It is because that the GT data are created by oversegmentation compared with the MS segmentations. The number of regions detected as oversegmention/undersegmentation in Dorin's method for gray images is larger than those of the proposed method. The total number of regions in the ideal case for six color images, 285, is also calculated



(a) oversegmentations (gray-level). (b) undersegmentations (gray-level).

Figure 4.21: Evaluation of over- and undersegmentations using ground truth for the gray-level image sequences.



(a) oversegmentations (color). (b) undersegmentations (color).

Figure 4.22: Evaluation of over- and undersegmentations using ground truth for the color image sequences.

based on the GT regions, not the MS regions (proposed: 320, Dorin: 391). Therefore, 35 (proposed) 106 (Dorin) regions in the MS regions are not participated in this evaluation (undetermined) respectively. The number of regions detected as oversegmention/undersegmentation in Dorin's method for color images is larger than those of the proposed method.

4.8 Conclusion

The neural networks based segmentation scheme which incorporates static and dynamic features simultaneously in one scheme was proposed in this Chapter. The soft feature weighting scheme based on the motion confidence measure was developed. Also, the edge fusion algorithm was also proposed to improve segmentation results. Use of edge linked maps during edge fusion creates more closed regions with segmentation boundaries closer to edges than the use of nonedge linked maps. The segmentation results of both gray-level and color image sequences were evaluated using evaluation metrics to show the utility of the proposed algorithm. The proposed feature integration scheme has a simple and parallel architecture with good segmentation results. The proposed schemes can be used for object representations which are essential for new emerging digital services.

Chapter 5

A VLSI Architecture for Image Sequence Segmentation

5.1 Introduction

Image and video data in a multimedia system contain higher volumes of data than other media such as audio and speech. By exploiting spatio-temporal redundancy in video data, a number of coding-oriented video data compression standards, for example, H.261 [25], H.263 [26] , MPEG-1 [23] and MPEG-2 [24], have been proposed for efficient transmission and storage applications. The MPEG-4 [9, 107, 108] and ongoing standard MPEG-7 [109], which are based on the object representation of image sequences by segmenting a scene, are targeted toward content-based manipulations with more coding efficiency.

Efficient VLSI implementations of complex video processing tasks for real-time applications are important, since new multimedia services mainly depend on efficient and low-cost hardwares executing the required high performance tasks [110, 111]. Conventional digital signal processors [112, 113] are highly focused on speech and audio signals and lack high performance capability for video signal processing. General-purpose processors targeted to PCs and workstations are adequate for high performance desktop processing. However, they are too expensive, power consuming, and

need an adaptation for the video signal processing. Hence, special architectural approaches for the high performance of video processing are required.

In general, there are two architectural strategies in video processor design: namely, dedicated and programmable. Dedicated (special-purpose) implementation is a direct mapping of a specific algorithm or groups of algorithms. Therefore, VLSI implementation of the individual hardware module to computational characteristics of the well-defined and fixed function results in highest efficiency in area and timing. In contrast, the programmable architecture executes different tasks under software control. The main characteristic of programmable architecture is the increased functionality suitable for large application tasks. However, programmable architecture does not use hardware resources fully and needs additional programming cost and hardware (memory), higher power consumption, and increased development time and cost. [114] and [115] give good surveys of the recent development of video processing processors.

MPEG-4 standard does not include a segmentation algorithm. It is assumed that segmentation information is available at the decoder side. Therefore, it is desirable to develop an efficient segmentation algorithm and its special-purpose architecture for real-time video applications.

Several dedicated approaches for image and video segmentation [116, 117, 118, 119] have been proposed in the literature. In [116] a VLSI architecture proposed for a dynamic scene analysis technique that can extract regions with motion. After thresholding gray-level difference values between two image frames, connected component analysis of the binary difference image is performed. By processing edge maps of the two frames and the labels from the connected component analysis, moving regions of a current frame are segmented. However, the outputs of this method mainly depend on the luminance difference motion features which are derived from simply subtracting two images and may contain noise, especially for textured image. Also, a

fixed number of motion detection processors, corresponding to the number of motion detected regions, limits a scene containing the number of the fixed motion detected regions.

In [117] a gray-level image segmentation hardware was presented by segmenting an image into variable-size rectangular blocks. This architecture was developed for segmentation-based video coding, and regions of rectangular blocks do not adequate for object-oriented applications requiring arbitrary shaped regions. A special-purpose VLSI architecture for the real-time segmentation of endoscopic images was proposed in [118] for segmenting the darkest region of the images. The architecture is based on pipelined implementation of an adaptive thresholding technique. The architecture is to detect the gastrointestinal lumen regions and generate binary segmented regions. Therefore, this technique cannot apply for the segmentation of real-life scene.

A real-time gray-level image segmentation hardware to assign image pixels to a few user-selected classes was proposed in [119]. It consists of a feature extraction and decision analysis part. The feature extraction part extracts 14 local and global descriptions of the image pixels. The decision part classifies the image pixel with one out of predefined objects based on those descriptions. This architecture was developed for inspection applications based on a supervised pattern matching, therefore, is not suitable for general-purposed applications for segmentation.

A multiple features/neural network based segmentation algorithm [102, 103] which provides information of a region or groups of regions for object-oriented video applications has been proposed in this research. It incorporates static and dynamic features simultaneously in one scheme for segmenting a frame in an image sequence. The proposed method can also segment image sequences with textured feature efficiently. The multiple features extracted from two consecutive image frames are integrated by SOFM neural networks [95]. In order to generate precisely located object boundaries, an iterative edge fusion process [103, 104] is applied.

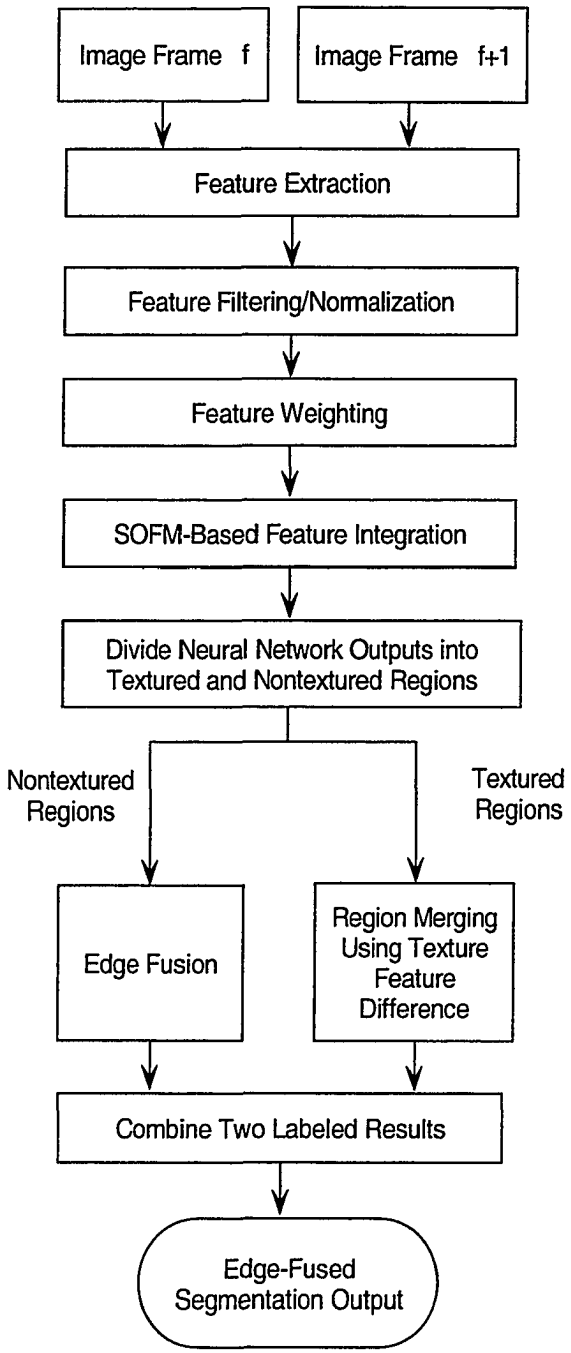


Figure 5.1: The proposed segmentation flow diagram.

The segmentation flow of the proposed scheme is shown in Fig. 5.1. Feature extraction generates feature vectors consisting of a gray-level or color features (three YUV color coordinates), a dense motion vector, and two simple texture features, for each pixel. Feature vectors are smoothed using median filtering (luminance/chrominance feature and motion features) and *min* filtering (two texture features). In order to remedy different dynamic ranges of the feature values, Mahalanobis distance [96] is used for feature normalization followed by feature weighting. These weighted features are integrated in SOFM neural networks in order to generate an initial label for each pixel. A linked edge map is incorporated into the nontextured regions of neural network outputs to generate more precisely located segmentation boundaries. A simple region merging process is applied for the textured regions using a texture feature similarity. The segmentation results for both nontextured and textured regions are combined to generate final segmented regions. In this Chapter, its function-specific VLSI architecture is focused and analyzed its time and hardware complexities for various image formats [120, 121].

The remaining parts of the Chapter are organized as follows: The segmentation system architecture is presented in Section 5.2. Performance analysis, and discussion and concluding remarks are given in Sections 5.3 and 5.4, respectively.

5.2 The VLSI Architecture

In general, image and video processing associate with high computation, and I/O bandwidths, and large storage. Image and video segmentation schemes involve complex subtasks performing different computational characteristics such as feature extraction, filtering, normalization, weighting, pixel labeling, and post-processing. The fundamental operations for segmentation can be classified as follows:

Pixel operations: The outputs depend on an input value at a pixel position. The input value can be a scalar or vector. For example, thresholding operation of a gray-level image uses an 8-bit input pixel data. In case multiple features are employed for segmentation, these pixel-based operations involve vector operations for each pixel. In general, thresholding, feature weighting, normalization, and labeling involve pixel operations.

Neighborhood operations: The outputs depend on the values of a current pixel and its neighbors. The feature extraction and filtering involve local neighborhood operations using a small window. Also, block matching motion estimation can be categorized in these operations. Pixel and neighborhood operations of image and video processing are highly regular tasks and can run in array processing such as systolic and pipeline architecture [122, 123].

Region operations: The outputs depend on the attributes (label, size, perimeter, gray-level/color means, etc.) of an arbitrary region, a set of pixels. These operations are typically used for region sorting, region splitting, and region merging. In general, these operations are iterative, since the results of a current operation should be fed into the next operation. Index addressing of a region label is needed for fetching a set of data in a region. Most post-processing operations of segmentation can be classified into this category.

An image sequence segmentation system involves complex subtasks. Therefore, the development of such system consists of the selection of appropriate methods in the literature and the development of novel systems. Figure 5.2 shows the block diagram of the entire segmentation system. In the frame buffers, three color components of the reference frame and a luminance component of the next frame are stored. From the two consecutive frames, the multiple features are extracted in parallel. Motion

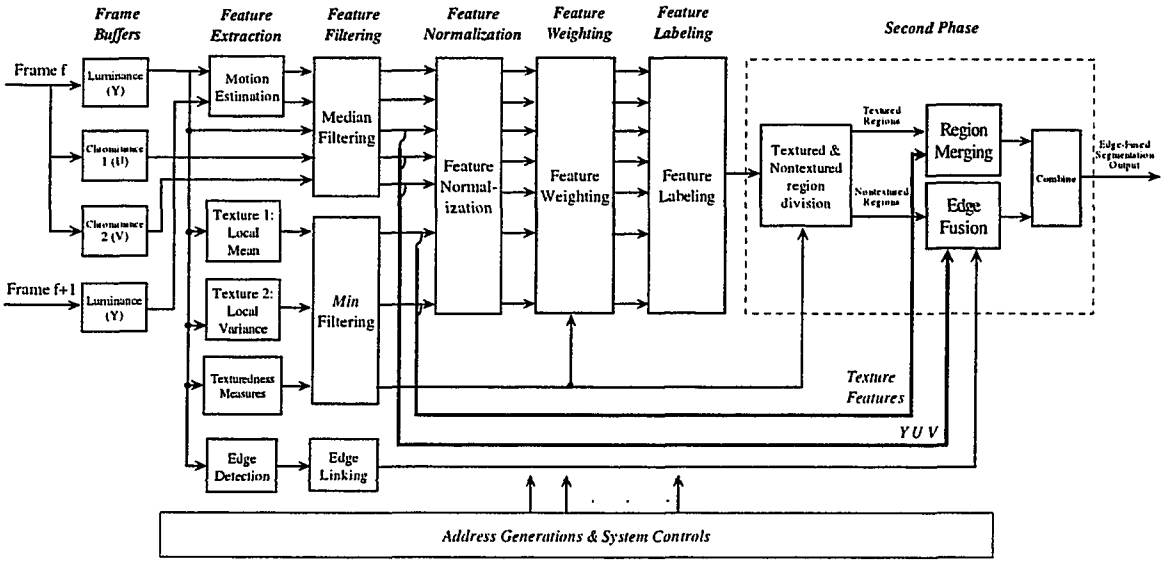


Figure 5.2: The block diagram of the entire segmentation system.

features of luminance components are estimated, and the two texture features and texturedness measures of the reference frame are extracted using local neighborhood operations. Also, edge detection and linking hardwares extract an edge map and its linked edge map for the edge fusion. A systolic median filter removes small details of motion and YUV features, and a *min* filter eliminates unwanted high values resulting from local neighborhood operations along region boundaries by choosing a small value within a window. The filtered features are normalized to have zero means and unit variance using parallel pipelined architecture, salient feature components of a feature vector are weighted high, and unreliable feature components are suppressed using a parallel pipelined feature weighting module. For the feature labeling, this weighted feature vector finds the closest index (label) of code words from the trained SOFM neural networks using systolic array architecture. The feature normalization, weighting, and labeling subtasks are run in a parallel pipelined manner. After dividing the labeled initial segmentation results into textured and nontextured regions based on the texturedness measures, edge fusion processing module refines oversegmented non-

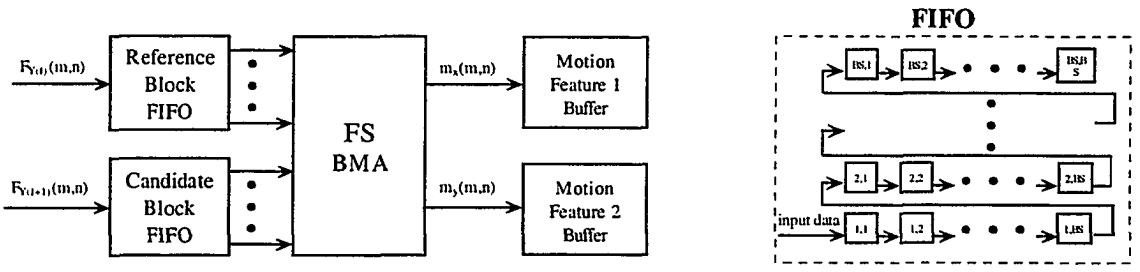


Figure 5.3: The block diagram of the motion estimation.

textured regions so that the region boundaries are closer to edge. Textured regions are merged using a texture feature similarity criterion. The two results are combined to generate a final segmentation. The address generation and system controller generates addresses for accessing data stored in the local buffers and controls the sequence of subtasks. In the following sections, each subsystem is described in more detail.

5.2.1 Feature Extractions

A feature vector consists of three YUV color components, a motion vector and two texture features. The block diagram of block matching motion estimation is shown in Figure 5.3. Let $B_s \times B_s$ and p be a matching block size and maximum displacement assumed, respectively. The 8 bit data for the reference block and candidate blocks are serially fed into $B_s \times B_s$ and $B_s \times (B_s + 2p)$ size FIFOs, respectively. The system controller generates addresses for scanning the data. A full search block matching architecture (FS BMA) based on mean absolute difference (MAD) arithmetic [124] is adapted. The systolic block matching arrays compute the best matching candidate block. The components of the motion $[m_x m_y]'$ are stored in the two motion feature buffers.

The number of clock cycles for the estimation of a motion vector is $(B_s+2p)(p+3)$. The motion estimation for the whole image needs $(N/({B_s}^2))(B_s + 2p)(p + 3)$ clock cycles, where N is the total number of pixels in an image. The estimated motion

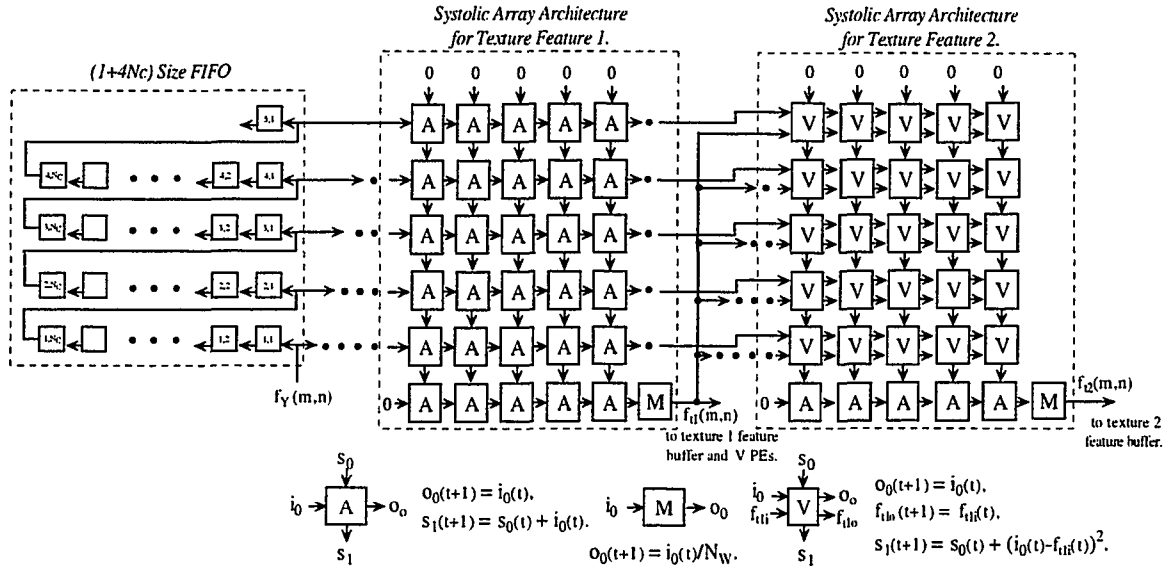


Figure 5.4: The systolic architecture for the texture feature extractions.

features are scaled up by 128 to be possible for 8 bit unsigned integer processing in the following subsystems.

The two texture features, local mean and variance, referred to as $f_{t1}(m,n)$ and $f_{t1}(m,n)$, respectively, are textural appearances of the area surrounding a pixel (m,n) in a small estimation window centered on this pixel, where N_w is the number of pixel of the size of $W_s \times W_s$ of window W as expressed in Equations 5.1 and 5.2.

$$f_{t1}(m,n) = \frac{1}{N_w}\sum\sum_{(k,l)\in W} f_Y(m-k,n-l), \tag{5.1}$$

$$f_{t2}(m,n) = \frac{1}{N_w}\sum\sum_{(k,l)\in W} (f_Y(m-k,n-l) - f_{t1}(m,n))^2, \tag{5.2}$$

where $f_Y(m,n)$ is a luminance value at pixel (m,n).

Figure 5.4 shows the systolic architecture for extracting the two texture features using a 5×5 sliding window. To feed the data into the systolic arrays, the luminance components of the reference frame $f_Y(m,n)$ are scanned and serially fed into the $1+4N_c$ size FIFO whose latency is $1+4N_c$ clocks. Let N_c be the number of columns of the reference frame. The two systolic arrays calculate a local mean and variance

every clock cycle with 5 clock cycle latency. For these operations, block A accumulates luminance components, and a final mean value is produced in block M by dividing the accumulated results by N_w. Also, original data from the FIFO and the mean texture feature are used as inputs for V blocks in order to calculate the local variance. The functionalities of A, M, and V blocks are shown in Figure 5.4.

The number of clock cycles to calculate the two texture features of an image is the summation of the latency of the FIFO, the two systolic arrays, and the size of the image subtracted by 1, i.e., $(W_s - 1)N_c + 2W_s + N$.

The texturedness measures $TM(m,n)$ used for feature weighting and division of nontextured and textured regions can be calculated using Equation 5.3. The $TM(m,n)$ is a $\min(\lambda_1, \lambda_2)$ of the matrix $Z(m,n)$ in Equation 5.3, where λ_1 and λ_2 are two eigenvalues of the matrix. The minimum eigenvalue is proportional to $\min\{\sum_{(k,l)\in W}(|f_{Yx}(m-k,n-l)|, |f_{Yy}(m-k,n-l)|)\}$, where f_{Yx} and f_{Yy} are gradient values of luminance components, f_Y in x and y directions, respectively. In order to simplify the computations, in this dissertation, Equation 5.4 for calculating the texturedness measure is used for the subsystem architecture.

$$Z(m,n) = \sum_{(k,l)\in W} \begin{bmatrix} f_{Yx}(m-k,n-l)^2 & f_{Yx}(m-k,n-l)f_{Yy}(m-k,n-l) \\ f_{Yx}(m-k,n-l)f_{Yy}(m-k,n-l) & f_{Yy}(m-k,n-l)^2 \end{bmatrix}, \tag{5.3}$$

$$TM(m,n) = \min\{\sum_{(k,l)\in W}(|f_{Yx}(m-k,n-l)|, |f_{Yy}(m-k,n-l)|)\}, \ (k,l) \in W. \tag{5.4}$$

The systolic array architecture for the texturedness measure using a 5×5 sliding window is shown in Figure 5.5. The $4N_c + 5$ size FIFO stores the luminance data of the reference frame. After a $4N_c + 5$ clock cycle latency, five pairs of x directional luminance values (f_{Yx}) of the left two columns are fed into the T blocks. Since luminance data in the frame are scanned row by row, 5 pairs of y directional luminance data in the 5×5 window are selected every clock using a MUX whose inputs are

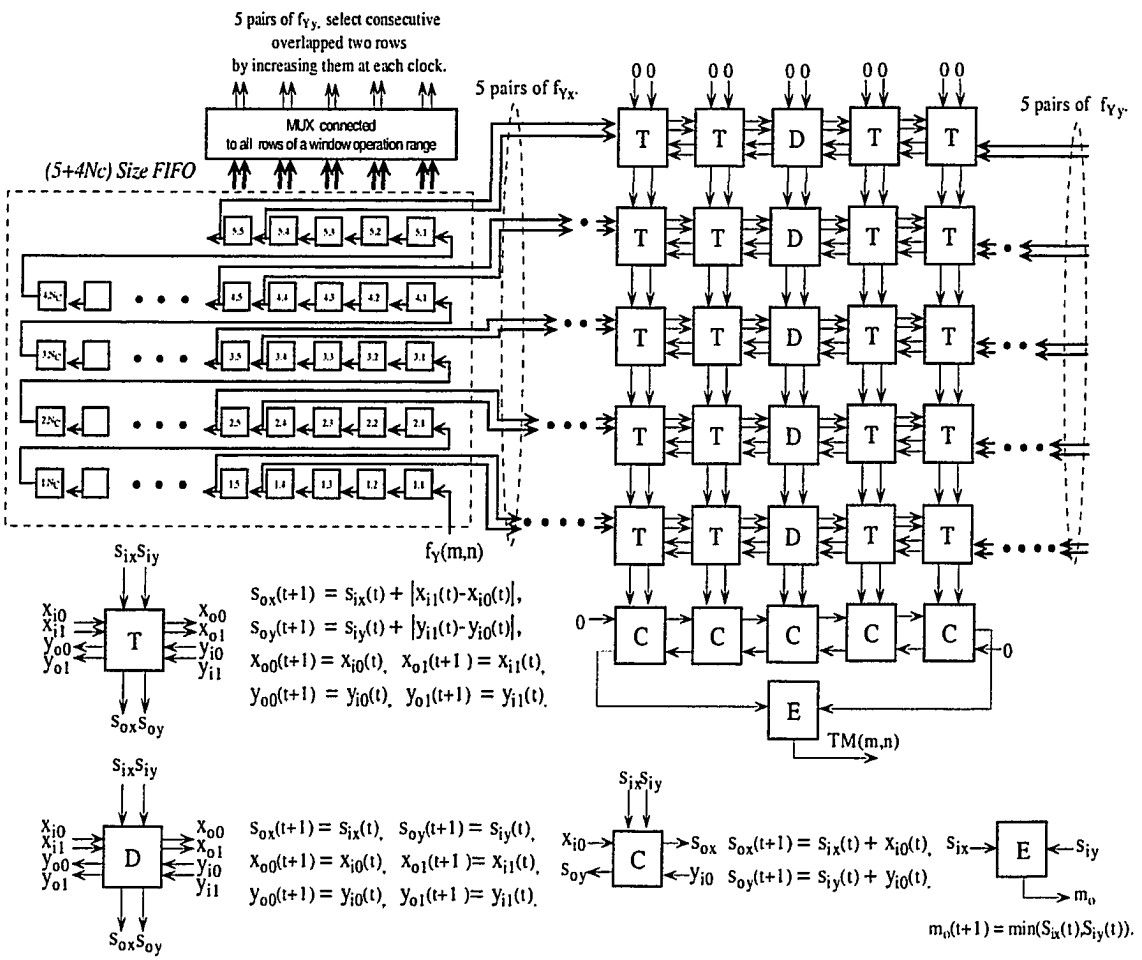


Figure 5.5: The systolic architecture for the texturedness measures.

connected to all the pairs of y directional luminance data within the 5×5 window operation ranges (the first two rows in the window are directly connected to the MUX, and one clock later, the second two rows (the second and third rows in the 5×5 local window) are shifted left or right). The $\sum f_{Yx}$ and $\sum f_{Yy}$ are computed simultaneously in the opposite direction using T and C blocks. The minimum value is chosen in the E block and saved in a texturedness measure buffer. The D blocks bypass their inputs with one clock delay. They are inserted for synchronization of the outputs of the systolic arrays. After 5 clock latency, the architecture generates $T(m, n)$ every clock.

The number of clock cycles is the summation of latency for the FIFO and the arrays and size of an image subtracted by 1, i.e., $W_s + (W_s - 1)N_c + W_s + N$. This module requires 8 bit $W_sN_c+W_s$ size FIFO, $W_s(W_s-1)$ T blocks, W_s D and C blocks, an E block hardwares, and a MUX to choose W_s pairs of f_{Yy} inputs. The variance texture features and texturedness measures are scaled to be 8 unsigned integers by storing just the first 8 MSBs for unsigned filtering operations.

Architectures proposed in [38, 39] are adapted for the edge detection and linking. The number of clock cycles required for edge detection and linking of an image is $3N$ and N, respectively. The edge detection hardware requires 650,000 transistors, and the edge linking hardware needs 11,520 FFs and 10,000 logic gates.

5.2.2 Feature Filtering

Three YUV color features and two motion features are applied for nonlinear median filtering, and the two texture features and texturedness measures are filtered using nonlinear *min* filtering. Median and *min* filters select a median value and minimum value after sorting the data in a small window. It is desirable to choose the nth smallest value instead of selecting the minimum value for the *min* filtering, because the minimum value makes the filtered outputs erode too much.

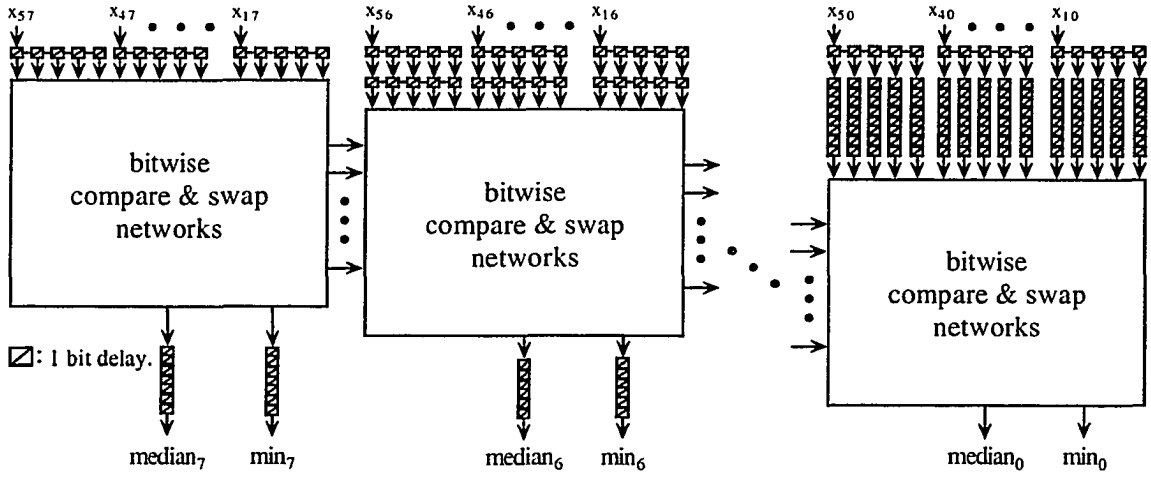


Figure 5.6: The bit-serial systolic architecture for the median and *min* filtering.

Figure 5.6 shows a bit-serial pipelined architecture similar to one in [125] for the median and *min* filtering using a 5×5 sliding window of 8 bit inputs. The bitwise compare and swap networks consist of compare and swap unit (CSU2) and bitwise delays. When the window is of size $W_s \times W_s$, the filter requires ${W_s}^2$ pipeline stages for each bit output. Each stage needs one bitwise delay and ${W_s}^2/2$ CSU2 modules. The 8 bit five feature data of the rightmost column in the 5×5 window, x_1, x_2, x_3, x_4 *and* x_5, are fed into the filter in parallel using the same $1 + 4N_c$ size FIFO as discussed in Section 5.2.1.

After ${W_s}^2 + 8$ clock latency, the filter produces a one full-word median or *min* value per clock cycle. Therefore, the number of clock cycles for 8 bit inputs for the filtering is ${W_s}^2 + W_s N_c + 8 + N$. Hardware required for filtering 8 bit inputs using a $W_s \times W_s$ window is a $W_s N_c + 1$ size FIFO, $8({W_s}^2({W_s}^2/2))$ CSU2 modules, and $8{W_s}^2 + ({W_s}^2 + 1)(8(8+1)/2)$ bitwise delays. A CSU2 unit consists of 21 combinational logic gates.

All the filtered feature components are stored in the feature filtered buffers so that they can be used for the feature normalization and edge fusion.

5.2.3 Feature Normalization and Weighting

Feature noramlization using Mahalanobis distance [96] can be expressed as follows:

$$\breve{f}_i(m,n) = \frac{f_i(m,n) - \overline{f_i}}{\sigma_i}, i = 1, \ldots, N_f, \tag{5.5}$$

where $\breve{f}_i(m,n)$ is a normalized feature i, $f_i(m,n)$ is the filtered feature i, $\overline{f_i}$ is a mean of feature i, N_f is the number of feature components, and σ_i is the standard deviation of feature i. Mahalanobis distance scales each feature so that all the features have a unit variance and zero mean.

After the feature normalization, feature weighting of all the feature components are followed as expressed in Equations 5.6 $\sim$ 5.8 and shown in Figure 4.6.

$$\tilde{f}_i(m,n) = \breve{f}_i(m,n) W_{lc}(TM(m,n)),\ i = \textit{ the YUV feature components}, \tag{5.6}$$

$$\tilde{f}_i(m,n) = \breve{f}_i(m,n) W_m(TM(m,n)),\ i = \textit{ the motion feature components}, \tag{5.7}$$

$$\tilde{f}_i(m,n) = 0.1\ \breve{f}_i(m,n),\ i = \textit{ the two texture feature components}, \tag{5.8}$$

where $\tilde{f}_i(m,n)$ is the weighted feature component i, $TM(m,n)$ is a texturedness measure at pixel (m,n), $W_{lc}(TM(m,n))$ (Equation 4.7) and $W_m(TM(m,n))$ (Equation 4.8) are weighting values for the color and motion feature components, respectively.

The pipelined architecture for the feature normalization and weighting is shown in Figure 5.7. From the seven filtered feature buffers, a feature vector of a pixel position (m,n) is scanned serially and stored in the registers of the first stage of the pipelined architecture. The means $(\bar{f}_i)$ and standard deviations (σ_i) of feature components i are system parameters precomputed from the feature components of the training image sequences and stored in the corresponding feature registers as shown in Figure 5.7. The dynamic ranges of normalized features are reduced because of the scale parameters, σ_i (for example, from the unsigned 8 bits to signed 5 bits).

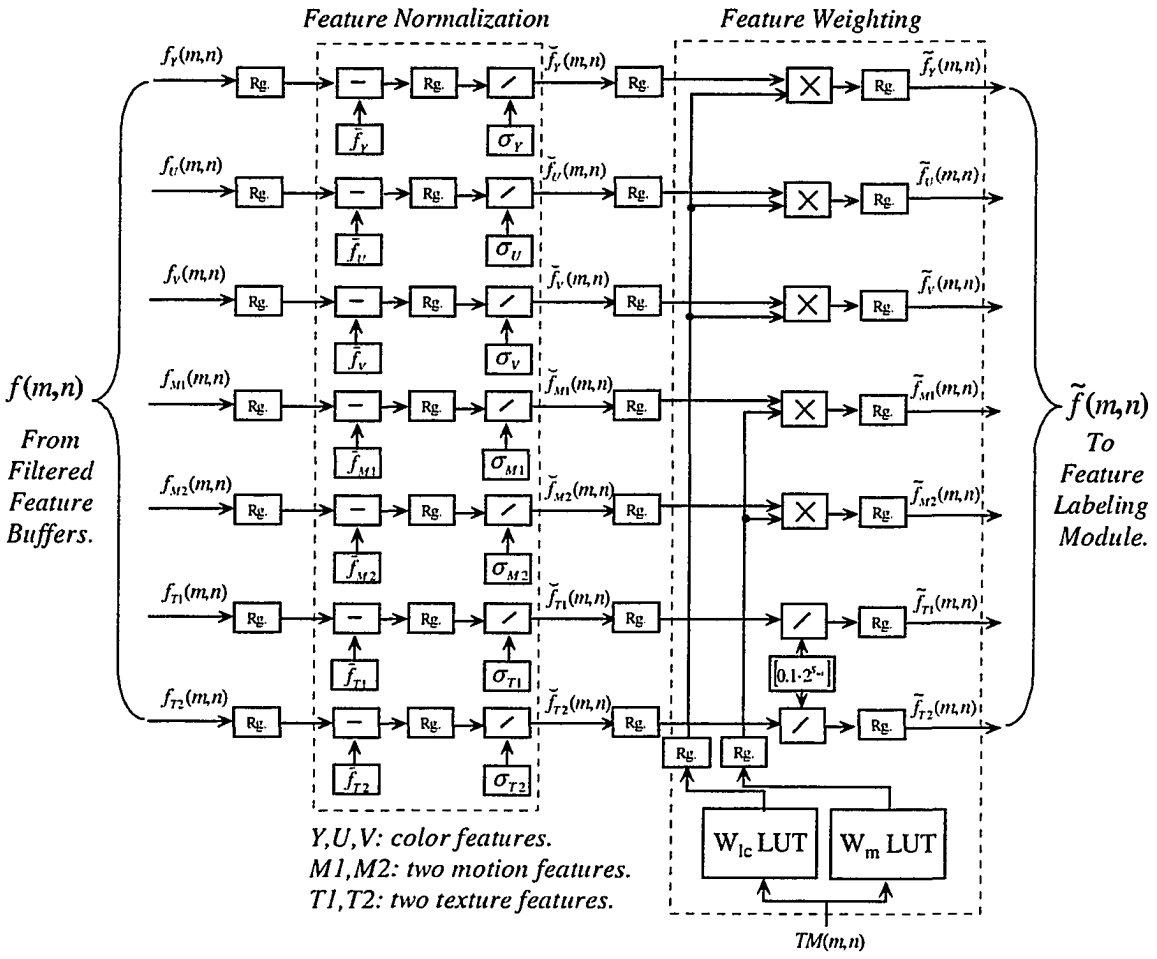


Figure 5.7: The pipelined architecture for the feature normalization and weighting.

In order to increase their dynamic ranges and resolutions, the precomputed σ_i can be scaled down by a factor of S_{nm} bits before stored in the registers (for example, scaled down by 3 bits to make signed 8 bits normalized results). This scaling to avoid float-pointing arithmetic is possible because we can also scale the trained code words for the next feature labeling by the same bits; its purpose is to find the best matching address (index) among the code words.

For the color and motion feature weighting, weighting functions $W_{lc}(TM(m,n))$ and $W_m(TM(m,n))$ are implemented using lookup tables (LUTs). These weighting values ranging from 0.0 to 1.0 are scaled up to be unsigned integers by a factor of S_{wt} (for example, 8 bits) and stored in the two LUTs. After the normalized data are available, the texturedness measure $TM(m,n)$ are read. The data outputs of a $TM(m,n)$ address input are multiplied with the normalized feautures and stored in the registers of the final pipeline stage. For the weighting of the two texture features, the normalized texture features are divided using $\lceil 0.1(2^{S_{wt}}) \rceil$ integer value. These weighted features are fed into the next systolic feature labeling module.

The throughput (bandwidth) of this module is one feature vector per clock cycle, and this module has 4 clock latency (4 pipeline stages). These feature normalization and weighting subtasks are operated in a pipelined manner with the feature labeling module. This module requires N_f normalization units, two $8 \times 2^{S_{wt}}$ size LUTs, (pipelined) registers, and multipliers (divisors) for the feature weighting. The word width of weighted features is increased by $S_{nm} + S_{wt}$ bits.

5.2.4 Feature Labeling

In order to generate an initial segmentation for a frame, a weighted feature vector of each pixel should have a label. The SOFM neural networks cluster training feature vectors into fixed code word vectors $\mathbf{c_j}$ $(1 \leq j \leq N_{cb})$ by incorporating neighborhood

neurons of a winning neuron into a competitive learning rule, where N_{cb} is the number of code words. The segmentation label of a pixel, $NL(m,n)$ is a segmentation label at pixel (m,n), i.e., an index of a code word vector which has a minimum Euclidean distance between an input feature vector $\tilde{\mathbf{f}}(\mathbf{m},\mathbf{n})$ and the code words expressed as follows:

$$NL(m,n) = \arg\min_j \{\|\tilde{\mathbf{f}}(\mathbf{m},\mathbf{n}) - \mathbf{c_j}\|^2 = \sum_{i=1}^{N_f} (\tilde{f}_i - c_{ji})^2\}, \tag{5.9}$$

where N_f is the number of feature components in a feature vector. The term $\|\tilde{\mathbf{f}}(\mathbf{m},\mathbf{n}) - \mathbf{c_j}\|^2$ can be transformed into inner product terms as follows:

$$\|\tilde{\mathbf{f}}(\mathbf{m},\mathbf{n}) - \mathbf{c_j}\|^2 = \|\tilde{\mathbf{f}}(\mathbf{m},\mathbf{n})\|^2 - 2(\tilde{\mathbf{f}}(\mathbf{m},\mathbf{n}) \cdot \mathbf{c_j} - 0.5\|\mathbf{c_j}\|^2), \tag{5.10}$$

where, $\tilde{\mathbf{f}}(\mathbf{m},\mathbf{n}) \cdot \mathbf{c_j} = \sum_{i=1}^{N_f} \tilde{f}_i(m,n) c_{ji}$. Since $\mathbf{c_j}$ is a fixed code word vector and $0.5\|\mathbf{c_j}\|^2$ can be precomputed, finding an index generating a minimum Euclidean distance is the same as searching an index producing a maximum $\tilde{\mathbf{f}}(\mathbf{m},\mathbf{n}) \cdot \mathbf{c_j} - 0.5\|\mathbf{c_j}\|^2$. Therefore, we can avoid the squaring operations for the feature labeling as follows:

$$NL(m,n) = \arg\max_j \{\tilde{\mathbf{f}}(\mathbf{m},\mathbf{n}) \cdot \mathbf{c_j} - 0.5\|\mathbf{c_j}\|^2\}. \tag{5.11}$$

The systolic architecture for the feature labeling is shown in Figure 5.8 when both feature components N_f and number of code word N_{cb} are 7. An input weighted feature vector $\tilde{\mathbf{f}}(\mathbf{m},\mathbf{n})$ is fed into this module with delays for synchronization of the output and shifted to the right to calculate the inner product through the entire code words. The L blocks accumulate the product of an input feature component and code words, and the results are shifted toward W blocks through S blocks which subtract $0.5\|\mathbf{c_j}\|^2$ from the inner product result. The W blocks search a minimum index (j) by comparing a current S block output (s_{i1}) and an temporal maximum value (s_{i2}) and its index (l_{ji}) shifted from the previous stage. A -2^B is a minimum B bit value assumed. If a current result from an S block is larger than that of the previous stage,

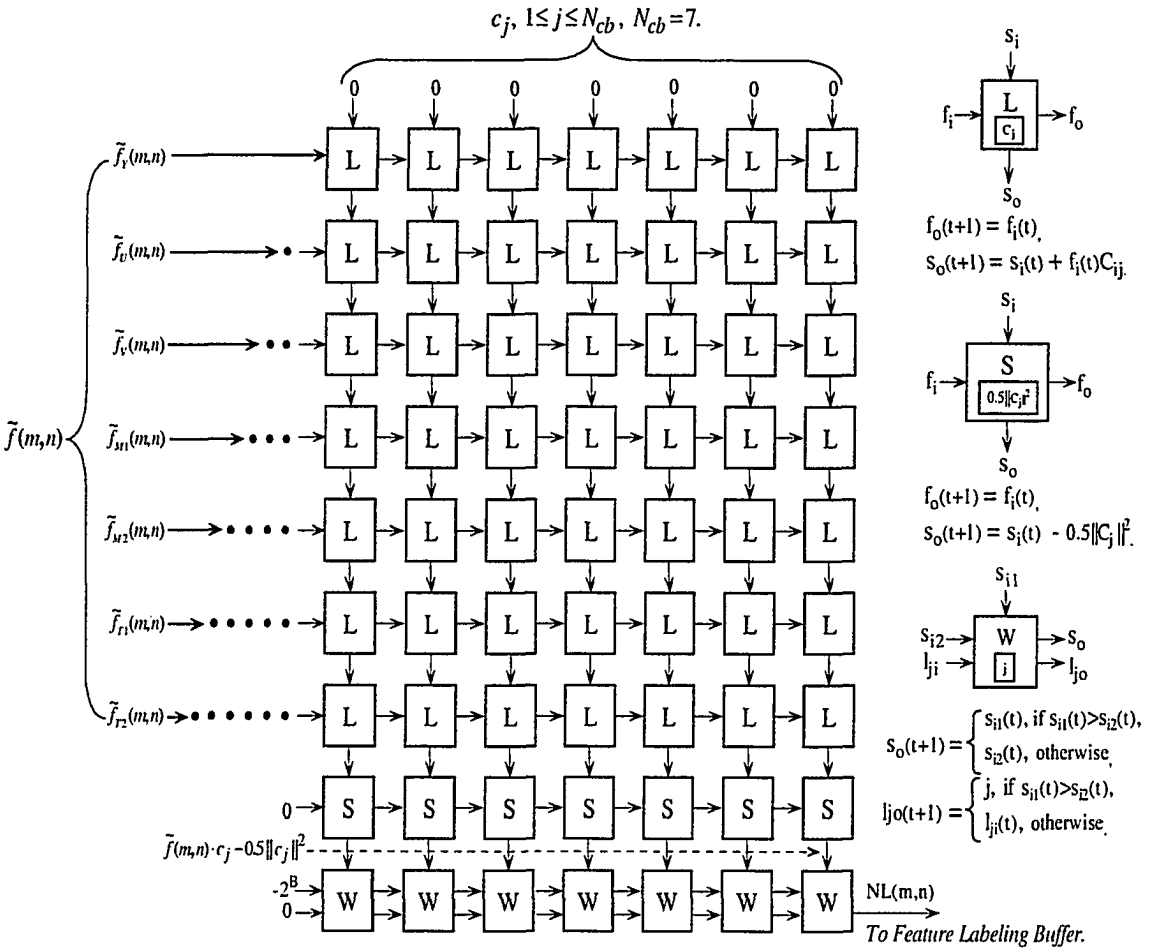


Figure 5.8: The systolic architecture for the feature labeling.

the W block swaps the temporal maximum value and its index for the current ones. Therefore, W block in the final stage can find an index of the best matching code word for the input feature vector, and the index $NL(m,n)$ is stored into a feature label buffer.

The throughput of this module is one feature vector per clock cycle, and its latency is $N_f + N_{cb}$ clock cycles. The feature normalization, weighting, and labeling are operated in pipelined manner. Therefore, the number of clock cycles for the three subtasks is $4 + N_f + N_{cb} + N$. This module requires $N_f N_{cb}$ L blocks, N_{cb} S and W blocks, and $N_f(N_F - 1)/2$ input delays.

5.2.5 Division of Textured and Nontextured Regions Splitting by Linked Edge

The oversegmented labels computed from the trained code words of the SOFM neural networks are merged to have lower segmentation labels through the second region merging process. First, the initial segmented regions are divided into textured and nontextured regions using the texturedness measure, $TM(m,n)$, and then the edge fusion process is applied to only nontextured regions after incorporating linked edges. Textured regions are merged by using the two texture features similarity. Figure 5.9 shows the division of the two regions and nontextured regions splitting by the linked edges.

The initial labels $NL(m,n)$ are fed into the FIFO to extract their contours (NLC in Figure 5.9) as described in Equation 5.12. After the contour extraction of the labels in order to fuse the binary linked edges, the data in three buffers (texturedness measures, the label contours (NLC), and linked edges (EL)) are scanned pixel by pixel in parallel and passed through the logic gates to generate edge-fused nontextured contours ($NTRC$) and textured region contours (TRC). The texturedness measure is simply thresholded by a predefined value TH_{tmeas} (BT in Figure 5.9). These

operations are expressed in Equations 5.13 $\sim$ 5.15. The edge-fused nontextured region labels ($NTRL$ in Figure 5.9) and textured region labels (TRL in Figure 5.9) are obtained by connected component analysis of the binary contours (Equations 5.16 and 5.17).

$$NLC(m,n) = \begin{cases} 1 & \text{if } NL(m,n) > NL(m-k,n-l), \\ 0 & \text{otherwise,} \end{cases}$$
$$(k,l) \in 4 \textit{ neighbors of } (m,n). \tag{5.12}$$

$$BT(m,n) = \begin{cases} 1 & \text{if } TM(m,n) < TH_{tmeas}, \\ 0 & \text{otherwise.} \end{cases} \tag{5.13}$$

$$NTRC(m,n) = \overline{BT(m,n)}NLC(m,n)EL(m,n). \tag{5.14}$$

$$TRC(m,n) = BT(m,n)NLC(m,n)EL(m,n). \tag{5.15}$$

$$NTRL(m,n) = CCA(NTRC(m,n)). \tag{5.16}$$

$$TRL(m,n) = CCA(TRC(m,n)). \tag{5.17}$$

The system controller scans the label buffer from the top left of an image to the bottom right row by row. To provide the data for the contour extraction, $2N_c + 2$ label values are stored in the FIFO. The scanned label data are serially fed into the FIFO. The total latency for producing the four neighbor labels is $N_c + 2$ clock cycles. If one of the neighbor labels (L_R, L_U, L_L, L_D) is larger than the current label (L_C), the scanned pixel has a binary contour value 0. The contour of each pixel of a row can be computed every clock cycle.

In order to generate region labels of the binary contours, a hardware architecture similar to one in [126] for the connected component analysis of the edge-fused nontextured binary contours and textured region contours is used. The nontextured region contours are analyzed first; textured ones, next. As soon as all the nontextured region labels are available, the edge fusion process begins. The connected component analysis architecture uses two-pass procedures. In the first pass, a bracket table is

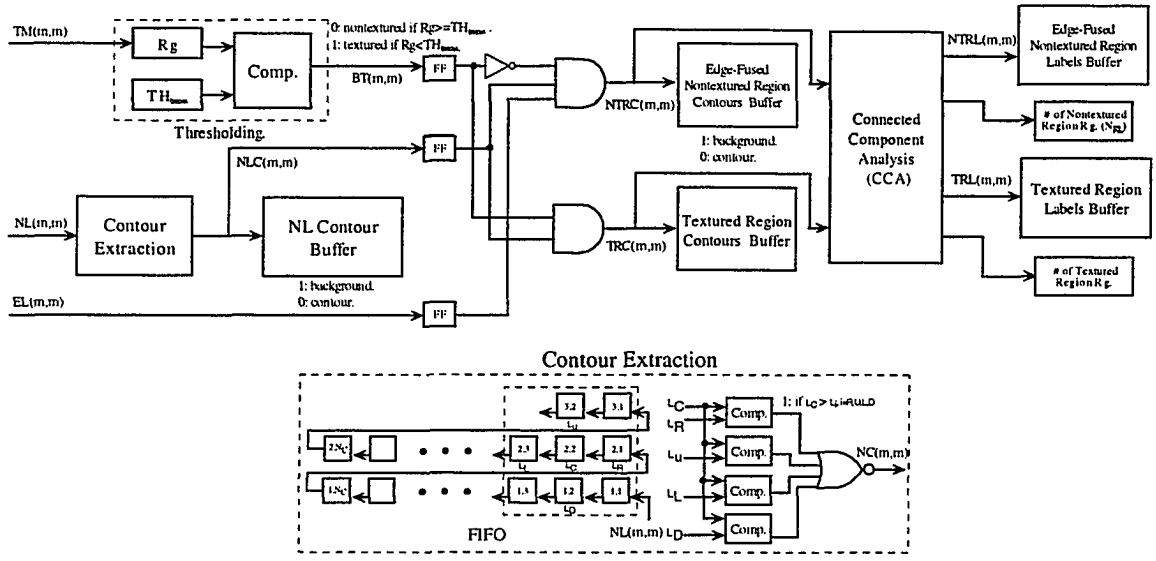


Figure 5.9: The division of textured and nontextured regions splitting by edge.

generated by scanning an image row by row and checking the set of runs. The connected component labeling is done in the second pass. This scheme needs three sweeps of memory. The connected component hardware can produce the number of regions of a binary input image used for the edge fusion and texture region merging. The hardware requires a pass 1 and 2 processors, MUXs, and an $N \times 1$ bit size bracket table.

The number of clock cycles (C_{NRS}) required for this subtask is the summation of that of contour extraction, region splitting by edge and connected component analysis of the two region contours, i.e., $C_{NRS} = (N_c + 1 + N) + N + 3N$. $2N_c + 2$ size FIFO, three image-size temporary contour buffers, and two output label buffers are required for this module.

5.2.6 Edge Fusion

The edge fusion algorithm integrates the edge linked map into the nontextured regions, and then iterates region merging processes until there is no region whose

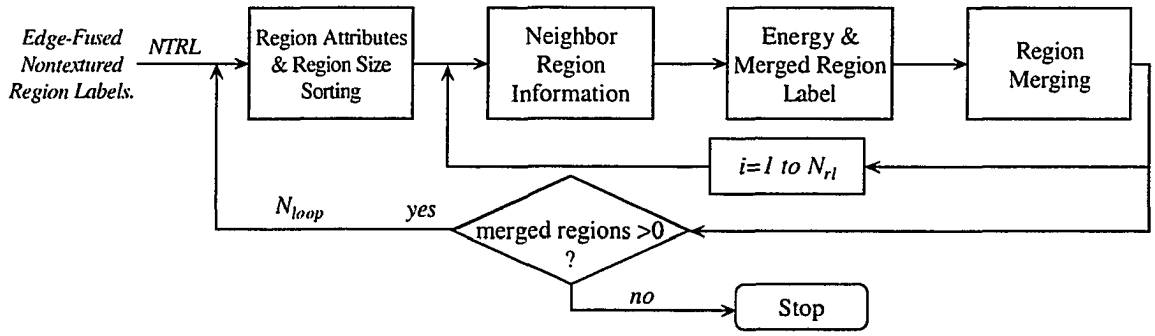


Figure 5.10: The flow diagram of the edge fusion.

luminance/color distance between neighbors is less than the segmentation resolution parameter α. As shown in Figure 5.10, the iterative edge fusion has an inner and outer loop. Let the maximum number of nontextured regions after overlapping the linked edge on the texuted regions be N_{rl}. After calculating region attributes and region size sorting, a region from the smallest region searches its neighbor information and a merged neighbor region label which has the smallest energy value. If a merged neighbor region exists, the current region and the neighbor region are merged. This step is continued for all the regions of the nontextured region labels buffer through the inner loop. If there is at least one merged region after the inner loop operations for all regions, the entire edge fusion procedure is iterated again through the outer loop until there is no merged region.

Figure 5.11 shows a simplified edge fusion architecture for color image sequences. The edge-fused nontextured region labels, edge linked map, and median filtered three YUV features have been stored in five local buffers. In the region attributes and sorting module, five region attributes (size, starting address, accumulations of YUV features) are computed for each region by accessing the label and YUV feature buffers. Also, the order of the region merging process is determined by sorting the region sizes of the labels. The order is stored in the sorted region label buffer. The region attributes for the current region label are stored in the registers which are accessed by

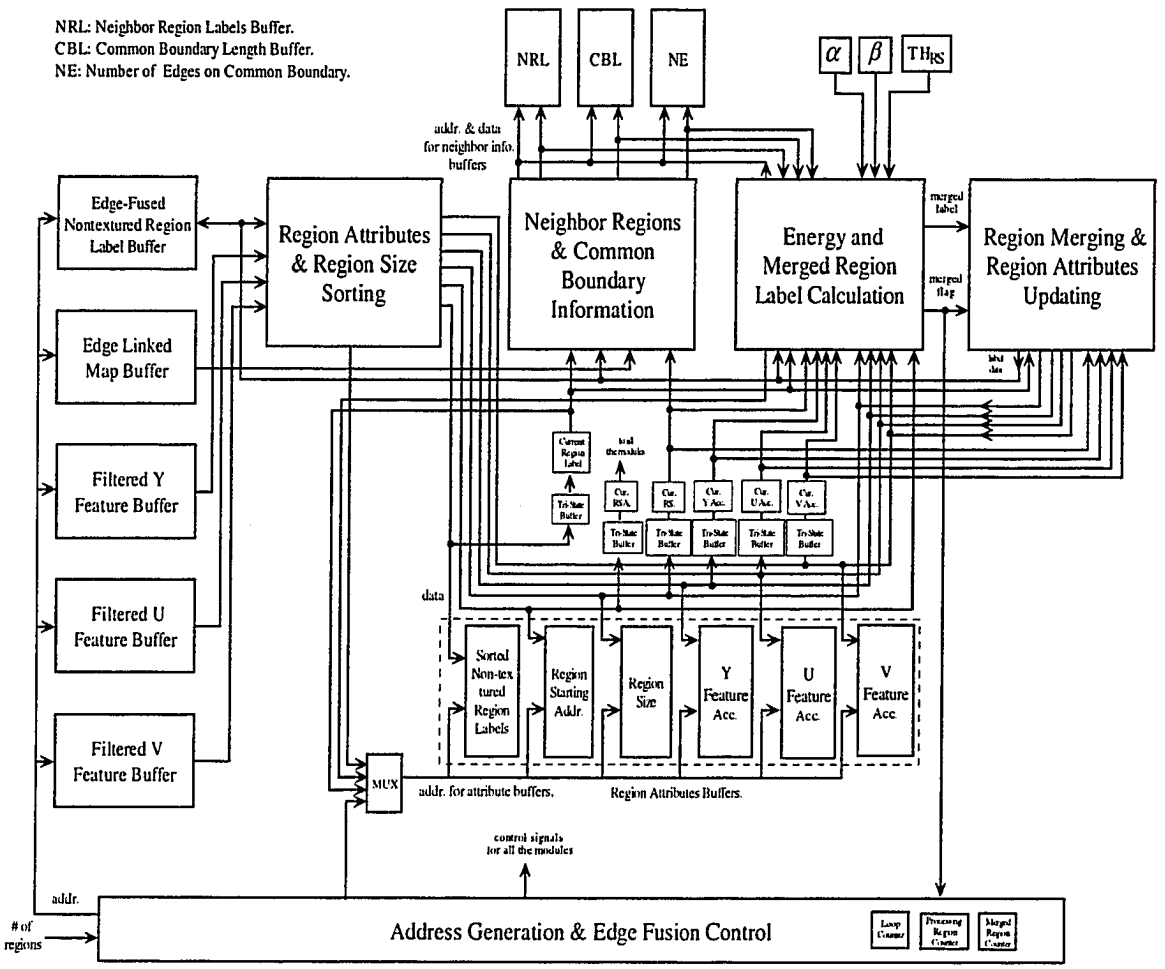


Figure 5.11: The simplified block diagram of the edge fusion architecture.

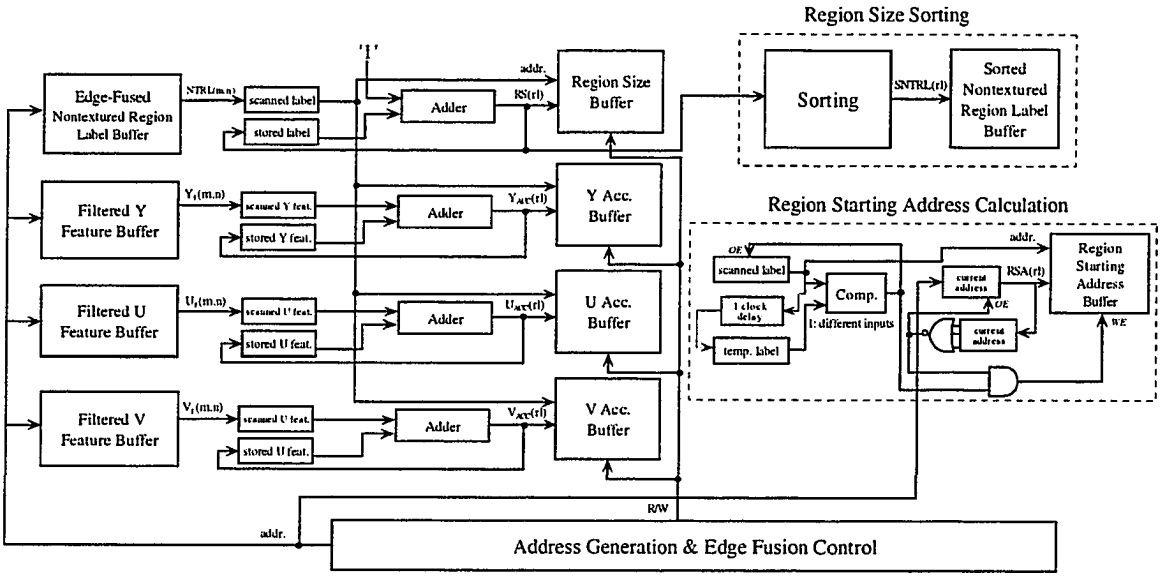


Figure 5.12: Region attributes and region size sorting.

subsystems. The neighbor region labels, common boundary lengths, and the number of edges on the common boundaries of the current label are stored in the three buffers connected to the subsystem called the neighbor region and common boundary information calculation module. With this neighbor information, a neighbor, which has a minimum energy, is determined using parameter registers, α and β in the energy and merged region label module. The current region is merged with the selected neighbor region. These operations are controlled by the address generation and system control module. The controller determines when to stop the region merging process using the merged region counter or a maximum allowable number of outer loop (N_{loop}). The details of each subsystem are described in the following subsections:

5.2.6.1 Region Attributes and Region Size Sorting

The order of labels for region merging is determined by region sizes, the smallest to the largest. The energy values between a current region and its neighbors determine whether a current region can be merged or not. For this reason, region attributes such

as size and YUV feature distances should be computed. Also, in order to reduce the indexed addressing time of a region, a starting address for each region is calculated and used for a preloaded addressing value of a region label during the addressing.

Figure 5.12 shows the process of region attributes calculation and region size sorting. The scanned label and three color feature values stored in the registers are accumulated (RS, Y_{ACC}, U_{ACC}, and V_{ACC} in Figure 5.12) and written in the corresponding region attribute buffers as described in Equations 5.18 and 5.19. At the same time, region starting addresses are calculated (RSA in Figure 5.12) using Equation 5.20. The region sizes are sorted in ascending order ($SNTRL$ in Figure 5.12) and written in the sorted nontextured region label buffer (Equation 5.21).

$$
\begin{aligned}
RS(rl) &= \Sigma p(m,n), \\
&\quad p(m,n) = \begin{cases} 1 & \text{if NTRL(m,n)} \in \text{rl}, \\ 0 & \text{otherwise}, \end{cases} && (5.18) \\
(Y,U,V)_{ACC}(rl) &= \{\Sigma(Y,U,V)_f(m,n) | NTRL(m,n) \in rl\}, && (5.19) \\
RSA(rl) &= \min\{(m,n) | NTRL(m,n) \in rl\}, && (5.20) \\
SNTRL(rl) &= sort\{RS(rl) | RS(SNTRL(1)) <, \ldots, \\
&\quad < RS(SNTRL(N_{rl}))\},\ rl = 1, \ldots, N_{rl}. && (5.21)
\end{aligned}
$$

The four input buffers are scanned pixel by pixel in parallel. A scanned label from the nontextured region label buffer corresponds to the address of the region size and YUV feature buffers (index addressing). All the attributes buffers are reset initially. When this address is available, the data of the four attributes are read and stored in the stored label registers. The stored region size value is added up by 1, and the scanned and stored YUV feature values are added and written. If the maximum possible region size, N_{rs} is predefined as N, the region size buffer needs an $N_{rl} \times \log(N)$ bit memory, and the other three attribute buffers need $N_{rl} \times (8 + \log(N))$ bit memory each.

The region starting address calculation runs in parallel with the region attribute calculations. The temporary register and the region starting address buffer are reset initially. When the scanned label value and temporary register value are different, the scanned label is written into the temporary register. Only when the fetched region starting address is zero and the comparator's output 1, the current address is written using index addressing of the scanned label.

A $\log(N_{rl})$-processor heapsort architecture [127] which takes $O(N_{rl}\log(N_{rl}))$ times and requires $8 + log(N) + log(N_{rl})$ bit word inputs is adapted for the region size sorting, since the outputs of the region size sorting module should be the index of region size, i.e., labels. The number of clock cycles (C_{RA}) required for this module is $C_{RA} = N + N_{rl}\log(N_{rl})$.

5.2.6.2 Neighboring Regions Information Calculation

Neighboring region information of the current region is calculated and saved for finding a neighbor with a minimum energy value required in the next module. The neighboring region information includes neighboring region labels, common boundary lengths and number of edges between a current region and neighboring regions.

Figure 5.13 shows the process of neighboring region information calculation. A scanned region label is fed into the FIFO, and it is compared with the current region label and its four neighbors to calculate a neighbor region label (NRL in Figure 5.13) as described in Equation 5.22. At the same time, common boundary lengths and the number of edges for between the current region and the neighbor region are accumulated (CBL and NE in Figure 5.13). These operations are expressed in Equations 5.23 and 5.24.

Instead of scanning the entire label buffers, a current region starting address is preloaded into the address generator so that it starts generating addresses for the 3×3 window operations at the region starting address as shown in Figure 5.14.

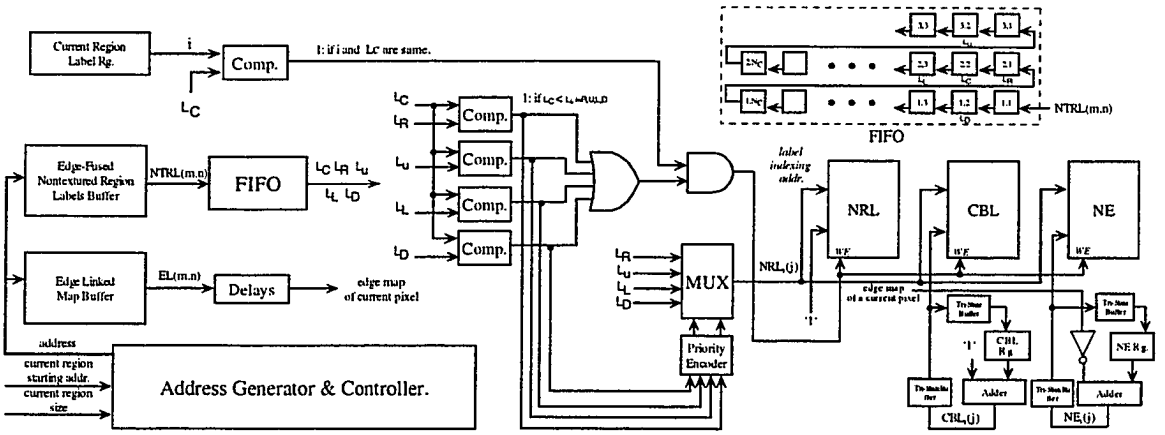


Figure 5.13: Neighbor region information calculation.

$$
\begin{aligned}
NRL_i(j) &= \begin{cases} 1 & \text{if } NTRL(m,n) == i \text{ AND} \\ & NTRL(m,n) < NTRL(m-k,n-l), \\ 0 & \text{otherwise,} \end{cases} \\
& (k,l) \in 4 \; neighbors \; of \; (m,n), \\
& j = one \; of \; NTRL(m-k,n-l), \\
& (m,n) \in \{pixels \; on \; the \; perimeter \; of \; label \; i\}. \qquad (5.22) \\
CBL_i(j) &= CBL_i(j) + 1, \; if \; NRL_i(j) == 1. \qquad (5.23) \\
NE_i(j) &= NE_i(j) + 1, \; if \; \{EL(m,n) == 0 | (m,n) \in \\
& \{pixels \; on \; the \; perimeter \; of \; label \; i \; whose \; neighbor \; is \; j\}\}, \\
& j = 1, \ldots, N_n \; (number \; of \; neighboring \; regions). \qquad (5.24)
\end{aligned}
$$

The address generation is stopped when a scanned label reaches the current region size. In this way, we can reduce the scanning time for calculating neighbors of an arbitrary region, since a region length ($RegL$), defined as the number of rows a region occupies, is generally smaller than the number of rows in an image, N_r. The same size FIFO as that in Figure 5.9 is used. If a scanned label L_C is the same as the current region label and one of the comparator's outputs using its four neighbors'

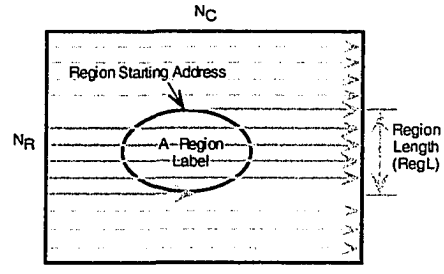


Figure 5.14: Illustration of a region length.

label inputs L_R, L_U, L_L, and L_D is logic '1', this binary value is stored (indexing address of the neighbor label) in the neighbor region label (NRL) buffer, and the fetched data (initially reset) of the corresponding common boundary length (CBL) and number of edge on the common boundary (NE) buffers are added by 1. The delays for the edge linked map's outputs are $N_c + 2$ clock cycles, which need to be synchronized with the scanned data for the current pixel label.

The memory requirements for this module are one $N_{rl} \times$ 1-bit memory for the $NRL_i(j)$, and two $N_{rl} \times \log(N_{rp})$-bit memories for both $CBL_i(j)$ and NE buffers, where N_{rp} is a possible maximum region perimeter (less than $6(N_r + N_c)$ in the estimate). The number of clock cycles (C_{NRI}) required of this subsystem is $C_{NRI} = (RegL+2)N_c + (N_c+2)$. The two rows ($2N_c$) should be added for scanning the label buffer using the 3×3 window.

5.2.6.3 Energy and Merged Region Label Calculations

Figure 5.15 shows the process of energy calculation and finding whether there is a merged neighbor region label or not between the current region and its neighbor regions. It consists of five pipeline stages (for the sake of simplicity, pipelined registers are not included in the figure). The feature distances of three color components (FD in Figure 5.15), the boundary strength between the current region label, and a neighbor region labels are computed ($S_i(j)$ in Figure 5.15) using Equations 5.25

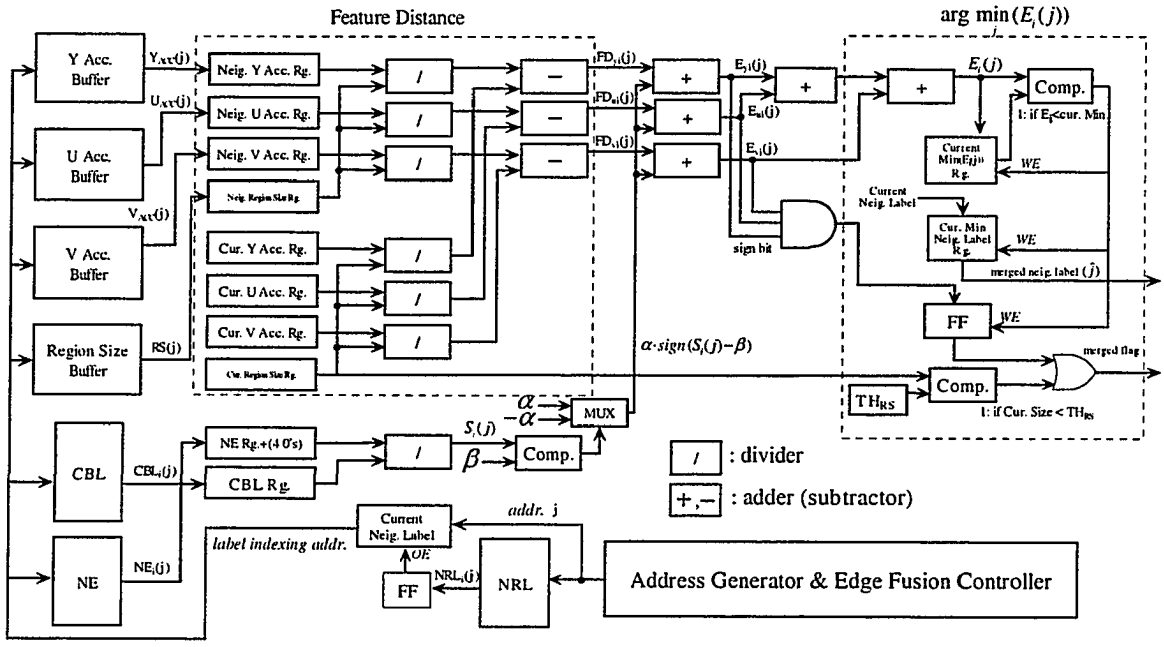


Figure 5.15: Energy and merged region label calculations.

and 5.26. The energy values of three color components are added (E in Figure 5.15) to find a merged neighbor region label as described in Equations 5.27 $\sim$ 5.29. The *merged flag* is set to be 1 only when all the energy values of three color components are negative and the current region is less than a predefined value as described in Equation 5.30.

$$FD_{(Y,U,V)i}(j) = \frac{(Y,U,V)_{ACC}(i)}{RS(i)} - \frac{(Y,U,V)_{ACC}(j)}{RS(j)}, \tag{5.25}$$

$$S_i(j) = \frac{16 * NE_i(j)}{CBL_i(j)}, \tag{5.26}$$

$$E_{(Y,U,V)i}(j) = FD_{(Y,U,V)i}(j) + \alpha \cdot sign(S_i(j) - \beta), \tag{5.27}$$

$$E_i(j) = E_{Yi}(j) + E_{Ui}(j) + E_{Vi}(j), \tag{5.28}$$

$$\hat{j} = \arg\min\{E_i(j)\}, \tag{5.29}$$

$$merged\ flag = \begin{cases} 1 & \text{if } (E_{Yi}(j), E_{Ui}(j) \text{ and } E_{Vi}(j) \text{ are all negative}) \\ & \text{OR } (RS(i) < TH_{rs}), \\ 0 & \text{otherwise,} \end{cases}$$
$$j = 1, \ldots, N_n. \tag{5.30}$$

The YUV feature accumulation values and the corresponding region sizes are read by index addressing (a label set as '1') of a region label changing from 1 (label 0 is regarded as a textured background label which cannot be merged with a nontextured current region label) to N_{rl} in the controller and stored in the corresponding registers. Feature distance values of YUV components between the current region and a neighbor region are added with the result of $\alpha \cdot sign(S_i(j) - \beta)$ to calculate an energy value $E_i(j)$, where i is the current region label and j is one of the neighboring region labels. The $S_i(j)$ is scaled up by 16 to be an integer. The α and β are predefined parameters. A neighbor region with a minimum value is updated in the $\arg\min_j(E_i(j))$ unit during iterative calculation of the energy values. After finishing energy calculations, the value of the current minimum neighbor label register (label $\hat{j}$) in the $\arg\min_j(E_i(j))$ unit is the neighbor region label which can be merged with the current region. In order to find negative energy values, the sign bits of the three YUV energy values are checked. Regardless of the energy values, if the current region size is less than a predefined threshold, TH_{rs}, the *merged flag* bit is set to be '1' so that the current small region is merged with the selected neighbor region. Also, this flag bit is connected to the controller, and the controller runs the region merging module only when the flag bit of the current region is set to be '1'.

The hardware mapping for the energy calculations can be easily done using dividers, comparators, adders, and MUXs. The number of clock cycles (C_{EC}) required of this module for each region is $C_{EC} = N_{rl} + 5$.

5.2.6.4 Region Merging and Updating Region Attributes

This module merges the current region with the neighbor region stored in the merged neighbor register and updates region attributes of the merged neighbor region label. The process of this module is shown in Figure 5.16. The region merging is the process of writing the merged neighbor region label $\hat{j}$ into the current region i as

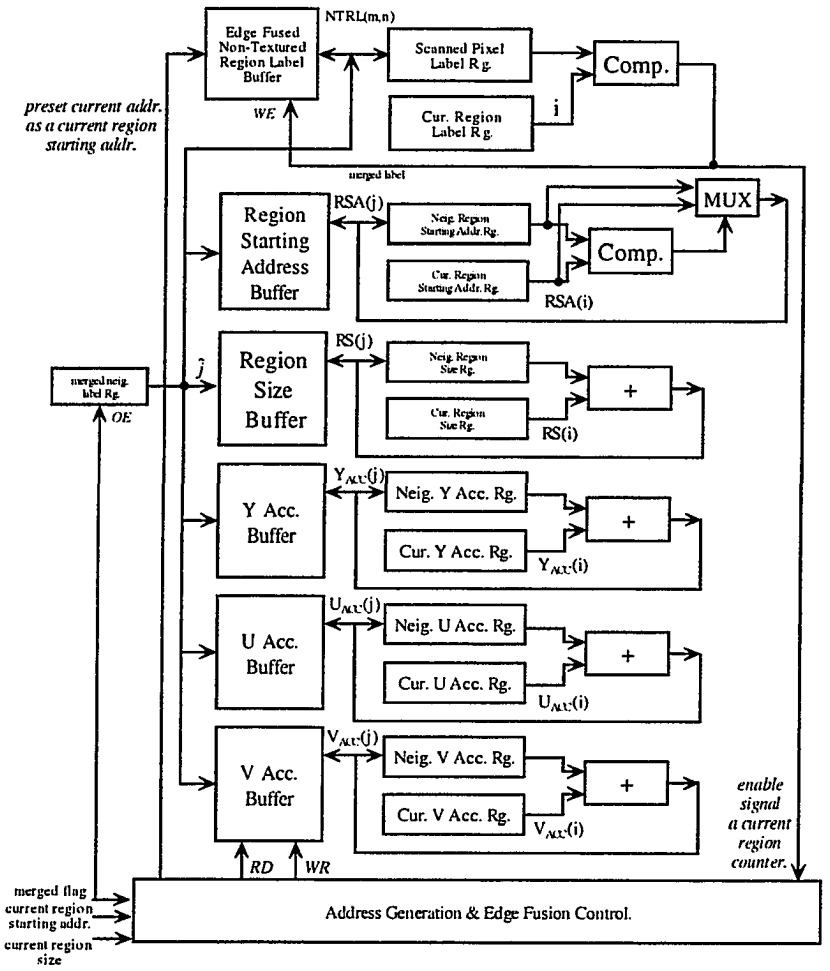


Figure 5.16: Region merging and updating region attributes.

described in Equation 5.31. The attributes (RSA, RS, Y_{ACC}, U_{ACC}, V_{ACC} in Figure 5.16) of the merged neighbor region are updated as expressed in Equations 5.32 $\sim$ 5.34.

$$NTRL(m,n)_{(m,n)\in i} = \hat{j}. \tag{5.31}$$

$$(Y,U,V)_{ACC}(\hat{j}) = (Y,U,V)_{ACC}(i) + (Y,U,V)_{ACC}(\hat{j}). \tag{5.32}$$

$$RS(\hat{j}) = RS(\hat{j}) + RS(i). \tag{5.33}$$

$$RSA(\hat{j}) = \min\{RSA(i), RSA(\hat{j})\}. \tag{5.34}$$

After the region merging, the corresponding region attributes for the merged neighbor region are updated for the next iterative merging process. Storing YUV feature accumulation values instead of the feature mean values makes the calculation of feature distance easier for the energy calculation, since updating YUV feature means needs many accesses to the YUV feature buffers. Updating the merged neighbor region attributes such as accumulated YUV feature values and region size is simply done by adding the attribute values of the current region with those of the merged neighbor region. For updating the region starting address for the merged region, a smaller value is stored in the merged label's region starting address, since the scanning of a buffer starts from the top of an image. The final segmentation for nontextured regions is stored in the $NTRL$ buffer.

The number of clock cycles (C_{RM}) for merging the two regions is $C_{RM} = (\textit{merged flag bit}) \cdot RegL \cdot N_c$, since when the *merged flag* is logic 0, this process is skipped.

5.2.7 Textured Region Merging and Generating Final Segmentation Result

The textured region merging subsystem, which runs in parallel with the edge fusion subsystem, has similar architecture to the edge fusion system but simpler, since it does not need edge information, and only the two texture features are used for a similarity measure for the textured region merging. If we replace the hardware components of the YUV color features in the edge fusion system with those of the two texture features and remove those regarding edge information, we can easily map the textured region merging system into hardware. In this dissertation, the more detailed explanations of the system architecture are skipped.

Figure 5.17 shows the process of the generation of a final segmentation. The contours of the merged nontextured and textured region labels ($NTRL$ and TRL) are extracted, and combined by ANDing operations (contour is logic 0 and background

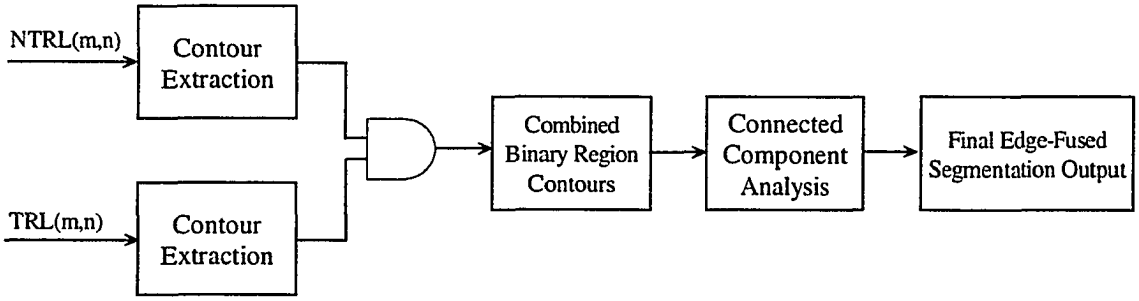


Figure 5.17: Combining of the merged nontextured and textured regions.

1). The final segmentation labels are generated by the connected component analysis of the combined binary contours. The number of clock cycles for this module is the summation of that of the contour extraction and connected component analysis, i.e., $(N_c + 1 + N) + 3N$.

5.3 Performance Analysis

The number of clock cycles and hardware components for the segmentation architecture are summarized in Tables 5.1 and 5.2. The hardware components are memory size, the number of registers, adders (subtractors), multipliers (dividers), and comparators. The overall complexity depends on an image size (N, N_r, N_c), the window size for neighborhood operations (W_s), the maximum number of feature components (N_f), and code words (N_{cb}). In addition, combinational logic gates are required for the filters, the edge detection subsystem, the edge linking subsystem, and the system controller. The temporary memory buffers can be shared to reduce the memory requirement.

The multiple features are extracted in parallel. Therefore, the number of effective clock cycles is that of a maximum feature extraction subtask between the motion estimation and the two texture feature extractions. Assuming that only one median and *min* filters are used for the system, five iterations of the median filtering (three

Table 5.1: The number of clock cycles of the VLSI architecture.

Subsystems	Clock cycles
Motion estimation	$(N/{B_s}^2)\ (B_s+2p)(p+3)$
Two texture features extraction	$(W_s-1)N_c+2W_s+N+1$
Texturedness measure	$(W_s-1)N_c+2W_s+N$
Median filtering per feature	${W_s}^2+W_sN_c+N+8$
Min filtering per feature	${W_s}^2+W_sN_c+N+8$
Feature normalization & weighting	latency: 4, throughput: one clock
Feature labeling	latency: N_f+N_{cb}, throughput: one clock
Region division	N_c+5N+1
Region attributes & sorting	$N+N_{rl}log(N_{rl})$
Neighbor region information	$N_c(RegL+3)+2$
Energy calculation	$N_{rl}+5$
Region merging	flag bit $\cdot N_cRegL$
Combine	$3N+N_c+1$

color and two motion features) and three iterations of the *min* filtering (two texture features and a texturedness measure) are required in order to generate a filtered feature vector. The edge detection and linking are run in parallel with the filtering and can be finished before the edge fusion process begins. Therefore, the number of clock cycles required for the filtering is five times that of one median filtering. The feature normalization, weighting, and labeling are done in a pipelined manner, and the throughput is one feature vector per clock cycle. The total number of clock cycles required for the iterative edge fusion is $C_{NRS}+N_{loop}(C_{RA}+N_{rl}(C_{NRI}+C_{EC}+C_{RM}))$.

To compare the performance of the proposed VLSI architecture to the equivalent MATLAB implementation for the same image sequences shown in Figure 4.17, Table 5.3 shows the parameter settings in the VLSI architecture. N_{rs}, N_{loop}, N_n, N_{rl}, N_{rp}, and $RegL$ are the worst-case parameters estimated during segmentation of the image sequences.

The number of clock cycles for each subtask estimated from the VLSI architecture for the color QCIF image segmentation is shown in Figure 5.18. If we compare

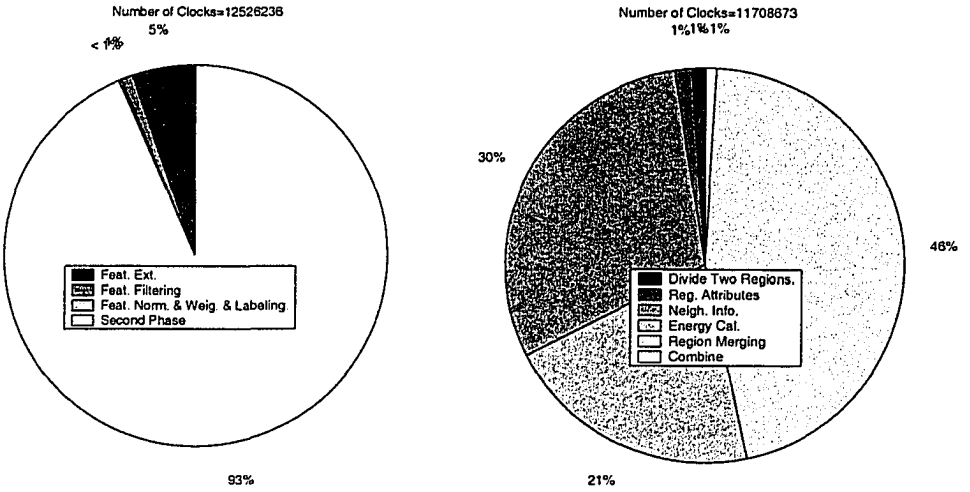
Table 5.2: The hardware components of the segmentation system.

Subsystems	Memory	Registers	Adders	Multi.	Comp.
Motion estimation	$2N \times 8$	$8{B_s}^2+$ $(9+2p)B_s+8$	$4{B_s}^2+$ $2B_s$	·	1
Two texture features extraction	$2N \times 8$	$7{W_s}^2+$ $(N_c+5)W_s+4$	$2{W_s}^2$	${W_s}^2$ $+2$	·
Texturedness measure	$N \times 8$	$8{W_s}^2-N_c+8+$ $(N_c+6)W_s$	$8{W_s}^2+$ $2W_s$	·	1
Median filtering per feature	$N \times 8$	$(44{W_s}^2+36)$FFs $+(W_sN_c+1)$ Rgs	·	·	·
Min filtering per feature	$N \times 8$	$(44{W_s}^2+36)$FFs $+(W_sN_c+1)$ Rgs	·	·	·
Feature norm. & weighting	·	$({N_f}^2+9N_f)/2$	N_f	$2N_f$	·
Feature labeling	$N \times log(N_{cb})$	${N_f}^2-N_f+$ $22N_{cb}$	N_fN_{cb} $+N_{cb}$	N_fN_{cb}	N_{cb}
Region division	$4N \times 1$, $2(N \times log(N_{cb})$	$9N_c/2$	·	·	5
Region attributes & sorting	$N_{rl} \times (5log(N_{rl})$ $+2logN+32)$	$\lceil 2log(N_{rl})+14 \rceil$	4	·	$\lceil log(N_{rl})$ $+1 \rceil$
Neighbor information	$N_{rp} \times$ $(2log(N_{rp})+1)$	$2N_c+20$	2	0	5
Energy calculation	·	35	8	7	3
Region merging	·	12	4	0	2
Combine	$N \times$ $(log(N_{rl})+1)$	$4N_c+4$	·	·	2

the CPU times estimated from the MATLAB implementation, the number of clock cycles for the edge fusion process is increased from 77% to 93% in the VLSI architecture. This is because the subsystems of the rest of the subtasks are implemented using systolic and pipelined architectures such that the processing speed is dramatically reduced. Therefore, the amount of the clock cycles is increased relatively for the subsystem of the iterative steps in the edge fusion process which cannot be implemented using pipelined architecture. In edge fusion, the subsystems for neighbor region information, energy calculation, and region merging, which are nested in both

Table 5.3: Parameter settings of the VLSI architecture.

Parameter	Value	Parameter	Value
B_s	4	N_{rp}	$6(N_r + N_c)$
N_{rs}	N	N_w	25
N_{cb}	49	p	12
N_f	7	$RegL$	3.57 per loop
N_{loop}	5	S_{nm}	3
N_n	N_{rl}	S_{wt}	8
N_{rl}	699 per loop	W_s	5



(a) entire system. (b) second phase.

Figure 5.18: The amount of clock cycles for the VLSI segmentation architecture.

inner and outer loops, account for 30%, 21%, and 46% of the edge fusion process, respectively, take most of the clock cycles.

Assuming using a modern ASIC technology and a clock rate of $400MHz$, Table 5.4 shows the processing time of the VLSI architecture for a variety of image formats. With estimated mean parameters, the processing times are reduced to 0.011, 0.033, 0.046, and 0.280 sec. for QCIF, 256 × 256, SIF, and CCIR 601 image types, respectively as shown in Table 5.5. The total segmentation processing time for a QCIF image with worst case parameters is 0.032 sec. This is five orders of magnitude

Table 5.4: The worst-case time analysis of the segmentation architecture.

Image Types	Number of clocks	Time (sec.) 400MHz
QCIF	1.2535e7	0.032
256x256	2.360e7	0.059
SIF	5.146e7	0.129
CCIR 601	35.011e7	0.876

Table 5.5: The time analysis of the segmentation architecture using estimated mean parameters.

Image Types	Average N_{rl} per loop	Average RegL	N_{loop}	Number of clocks	Time (sec.) 400MHz
QCIF	431	4.02	3.83	0.436e7	0.011
256x256	649	4.83	4.2	1.312e7	0.033
SIF	431	13.63	3.83	1.828e7	0.046
CCIR 601	431	54.82	3.82	11.187e7	0.280

improvement over the MATLAB implementation. For the detailed estimated data for the computational complexity, please refer to Appendix G.

The total hardware complexity for the entire segmentation system was estimated using the worst-case parameters and is summarized in Table 5.6. Local buffers are shared to reduce the memory requirement. The larger the image size, the more the memory requirement and registers. The numbers and average widths of other hardware components are not strongly dependent on the image size. For the detailed estimated data for the hardware complexity, please refer to Appendix H.

5.4 Conclusion

In this Chapter, a VLSI architecture for a novel image sequences segmentation algorithm has been proposed. The dedicated special-purpose system consists of motion estimation, edge detection, edge linking, median and *min* filters, feature normalization and weighting, systolic SOFM feature labeling, and edge fusion hardware. Compu-

Table 5.6: The hardware complexity estimation of the segmentation architecture.

Image Types	Memory (bit)	Registers (avg. width)	Adders (avg. width)	Multipliers (avg. width)	Comparators (avg. width)
QCIF	3.76 M	8121 (6.95)	659 (24.19)	402 (15.24)	99 (22.23)
256× 256	8.99 M	10529 (6.93)	659 (24.21)	402 (15.27)	99 (23.39)
SIF	12.87 M	10277 (6.95)	659 (24.22)	402 (15.30)	101 (22.44)
CCIR 601	52.52M	15471 (6.93)	659 (24.24)	402 (15.35)	103 (22.72)

tational and hardware complexities of the proposed architecture were estimated in terms of the number of clocks, arithmetic components, and memory requirements. The performance gain of five orders of magnitude over software implementation can be achieved. The performance of the proposed VLSI architecture demonstrates the possibility of performing MPEG-4-like object-oriented segmentation in real-time.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

A segmentation method which can provide regions or groups of regions for object-oriented segmentations of image sequences has been proposed.

The segmentation method uses a multiple feature clustering technique in Chapter 4. The feature vector consists of motion features and the static features such as luminance/chrominance and the two texture features. After filtering and normalization, only reliable motion features are selected using the soft feature weighting scheme. In this way, we can segment textured regions of an image sequence using low-dimensional texture features as shown in Figure 4.8.

Weighted features are clustered using the unsupervised SOFM neural networks in order to produce initial segmentation labels. The neural networks provide simple and parallel arithmetic operations, and they can be easily mapped into hardware. Since the trained cluster code vectors can be reused for the segmentation of all of the testing images, this method is faster than conventional iterative clustering algorithms.

The test images consists of gray-level and color image sequences with both simple and complex scenes and are divided into training and testing images. The same code weights in the trained neural networks are applied for the segmentation of all the testing images. Perhaps one of unique contributions of the proposed method is

edge fusion. The oversegmented initial segmentation results from the neural network outputs are merged in the edge fusion process to generate region boundaries closer to reality as shown in Figure 4.11.

The segmentation results for both gray-level and color image sequences were evaluated using intrinsic goodness measures and ground truth in order to evaluate the proposed segmentation method objectively. The proposed segmentation results are 68.2% (gray-level) and 8.4% (color) better in terms of FOM than those of Dorin's for real-life MPEG image sequences. The complexity analysis of the proposed method shows 22% faster processing time than the conventional K-means algorithm as shown in Figure 4.12.

Based on the proposed segmentation algorithm, a VLSI architecture has been proposed in Chapter 5. The subtasks before the edge fusion use regular arithmetic structure and are implemented using systolic and pipelined architectures. Therefore, the throughput of these substasks is high. Edge fusion needs an iterative region merging process and takes most of the clock cycles. In order to reduce the memory access time for the index addressing of a region label, the subsystem for region attributes calculates a region starting address of each region label before the merging process and utilize it as a region attribute during the edge fusion process. Instead of calculating feature means for updating region attributes after merging, this research took the approach of using accumulated region feature values and region size. Updating is done by adding the two feature values with those of a merged region; actual feature mean computation using updated accumulated features is done by a divider in the energy computation module.

The performance and system complexity of the VLSI architecture were estimated using the worst-case parameters extracted during the segmentation experiments of the MPEG image sequences. Every frame of the QCIF image sequences can be segmented

in real time even with the worst-case parameters (0.032 sec.). An SIF image used for video telephony applications can be segmented in 0.129 sec. with the worst-case parameters (0.046 sec. with mean parameters). Since image sequences have strong temporal correlations in general, all the applications do not need segmentation of every frame. If an application needs segmentation for every f frame, where $f = 4$ (0.129/0.033) for SIF image sequences at the worst case, the proposed system can be applied for the real-time segmentation of the SIF image sequences ($f = 1.4$ for average parameters). The performance gain of five orders of magnitude (15875) over the MATLAB implementation is achieved. The estimated amount of hardware is modest, and it is economically feasible to have a chip-set solution using modern VLSI technology.

6.2 Future Work

Our research has been focused on the image sequence segmentation algorithm and its VLSI architecture. The segmentation outputs provide more meaningful region information for object-oriented applications of image sequences such as MPEG-4 coding and content-based manipulation, since the multiple features including edge information are incorporated. The proposed scheme generates a homogeneous static region when the estimated motion vectors of the region are unreliable, and when several regions have homogeneous reliable motion features, the proposed algorithm merges the regions to create a segmentation which better presents the objects in the image. However, the proposed scheme has limitations and can be further improved in the future:

- During the SOFM neural network training for clustering feature vectors, the feature vectors on edge points are included as training feature vectors. These

features can be regarded as outliers of the clusters. Therefore, a feature clustering excluding these outliers can lead to better segmentation results.

- The initial labeled regions from the neural network outputs are broken into textured and nontextured regions before the edge fusion process begins. For this reason, a binary region mask is generated by simply thresholding the motion confidence values (texturedness measure). In some images, the small patches of a spurious binary regions are unintentionally included in the edge fusion process even if the *min* filtering is applied to the texturedness measures before the generation of the binary region mask. These spurious regions can be removed by combining another texturedness criteria such as an edge busyness and the energy difference of each subband using wavelet decomposition of the frame.

- The proposed edge fusion architecture needs a lot of memory access of the region label buffer to calculate neighboring region information of an arbitrary shaped region. In order to reduce the memory access time, the proposed architecture calculates the starting address of each region and uses it as a preloaded input for an address generation counter. Since the order of merged regions is decided by sorting the labeled regions before the iteration, the neighbor region information for the next iteration can be calculated during operations of other subsystems if we use two parallel neighbor region calculation modules which do not require much hardware complexity.

Appendix A

Abbreviations and Acronyms

ASIC	Application-Specific Integrated Circuit
BPN	Back-Propagation neural Networks
CIF	Common Intermediate Format
CCIR	International Radio Consultative Committee
DCT	Discrete Cosine Transform
DPCM	Differential Pulse Code Modulation
DSBV	Double Stimulus Binary Vote Method
DSCQS	Double Stimulus Continuous Quality Scale Method
DSIS	Double Stimulus Impairment Scale Method
FIFO	First In First Out
FS BMA	Full Search Block Matching Algorithm
GRF	Gibbs Random Fields
GT	Ground Truth
HVS	Human Visual System
ITU	International Telecommunication Union
ISDN	Integrated Services Digital Networks
ISO	International Organization for Standardization
LAN	Local Area Networks
LUT	Look Up Table
MAD	Mean Absolute Difference
MAP	A maximum a posteriori
MF	Motion Failure
MPEG	Moving Picture Experts Group
MRF	Markov Random Field
MSE	Mean Square Error
MS	Machine Segmentation
MUX	Multiplexer
NTSC	National Television System Committee
PAL	Phase Alternating Line

PDF	Probability Density Functions
PSTN	Public Switched Telephone Network
QCIF	Quarter Common Interface Format
RSST	Recursive Shortest Spanning Treee
SECAM	Sequential Couleur Avec Memoire
SIF	Source Input Format
SNR	Signal to Noise Ratio
SOFM	Self-Organizing Feature Map
SS	Single Stimulus Method
VHDL	VHSIC (Very High Speed Integrated Circuit) Hardware Description Language
VLSI	Very Large Scale Integration
VO	Video Object
VOP	Video Object Plane

Appendix B

Notations

(m, n)	a pixel position in an image
$(Y, U, V)_{ACC}$	accumulated Y, U, V features
α	segmentation resolution
β	threshold for region boundary strength
σ_i	variance of feature i
B_s	motion estimation block size
$BT(m, n)$	binary texturedness value at (m, n)
CB_{ij},$CBL_i(j)$	common boundary length between region i and j
C_{ij}	connectivity measure
$\mathbf{c}_j$	jth code word in a code book
$\mathbf{d}$	displacement, motion vector
ED_{ij}	number of edge between region i and j
E_{ij},$E_i(j)$	energy value between region i and j
$EL(m, n)$	linked edge value at (m, n)
$\mathbf{f}(m, n)$	feature vector at (m, n)
FD_{ij}, $FD_i(j)$	feature distance between region i and j
$\bar{f}_i$	mean of feature i
$f_{Y(f)}(m, n)$	Y component value of frame f at (m, n)
$I(x, y)$	intensity value at (x, y)
$\hat{j}$	a merged neighbor region label
N	number of pixels in an image
λ	an eigenvalue
N_c	number of columns in an image
$NC(m, n)$	neural network contour value at (m, n)
N_{cb}	number of code words in a code book
$NE_i(j)$	number of edge points between region i and j
N_f	number of feature components in a feature vector
$NL(m, n)$	feature labeled value at (m, n) after feature labeling
$NLC(m, n)$	a contour value of a neural network label at (m, n)

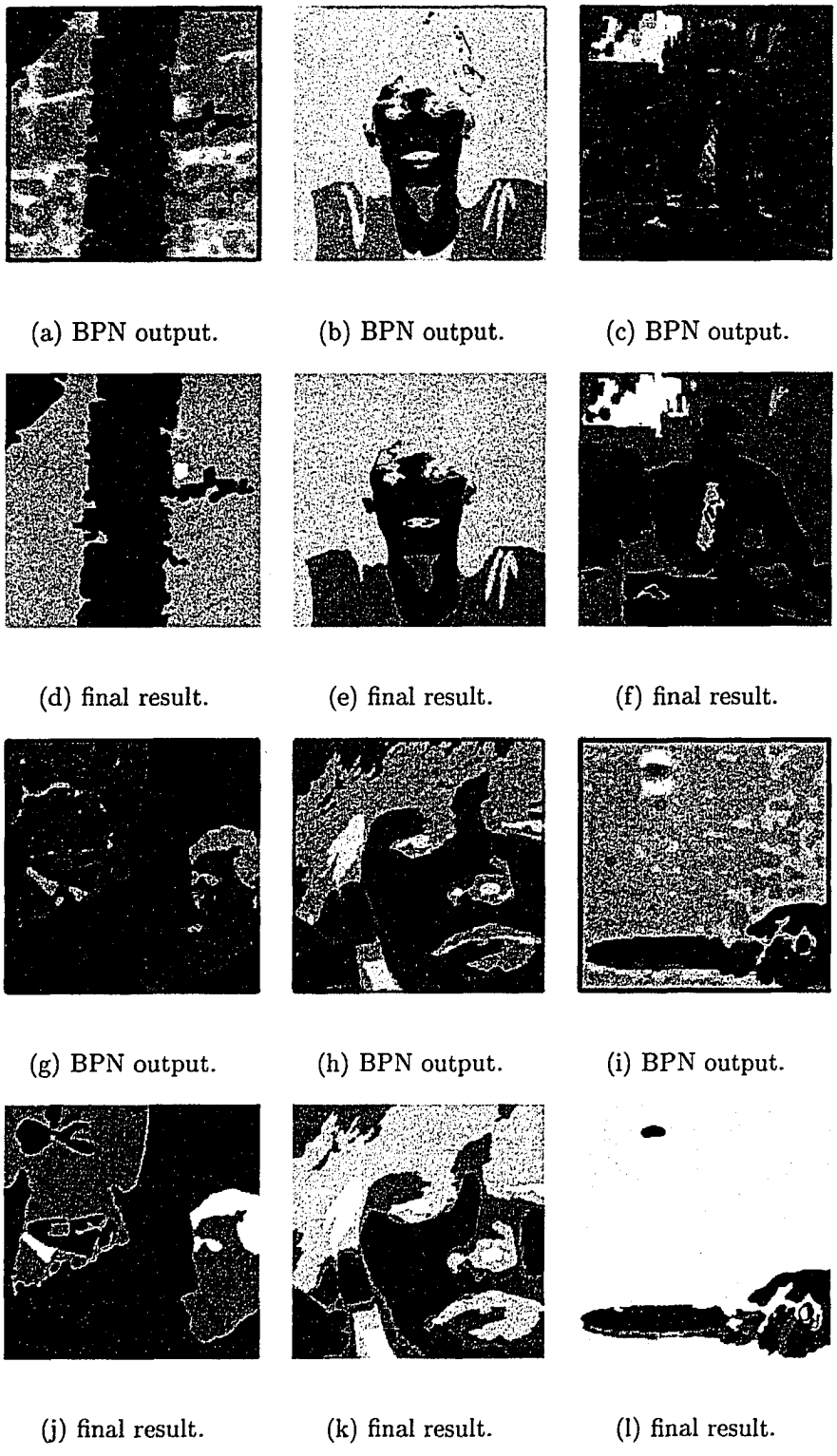
N_{loop}	number of loop during edge fusion
N_{merg}	number of merged regions during edge fusion
N_n	number of neighbor regions
N_r	number of rows in an image
N_{rl}	number of labeled regions
$NRL_i(j)$	neighbor region label, j of region i
N_{rp}	region perimeter
N_{rs}	maximum region size
$NTRC(m,n)$	nontextured region contour value at (m,n)
$NTRL(mn)$	nontextured region label at (m,n)
N_w	number of pixel in a local window W
p	displacement for motion estimation
P_i	perimeter of region i
$RegL$	a region length
R_m	region m
$RS(rl)$	size of region rl
$RSA(rl)$	starting address of region rl
$sign$	sign function
S_{ij}, $S_i(j)$	boundary strength between region i and j
S_{nm}	scaled bits for feature normalization subsystem
$SNTRL(rl)$	sorted nontextured label index of region rl
S_{wt}	scaled bits for feature weighting subsystem
TH_{rs}	threshold value for removing small regions
TH_{tmeas},TH_{eig}	threshold value for texturedness measures
$TM(m,n)$	texturedness measure at (m,n)
$TRC(m,n)$	textured region contour value at (m,n)
$TRL(m,n)$	textured region label at (m,n)
W	a local window
$W_{lc}(x)$	weighting function for luminance/chrominance feature
$W_m(x)$	weighting function for motion feature
W_s	local window size
Z	matrix for calculating motion confidence

Appendix C

Segmentation Results Using Back-Propagation Neural Networks [1]

The parameters for the segmentation are as follows:

- Image size: gray-level 256 × 256.
- Local window size: 5 × 5.
- Feature Weighting: hard weighting,
  10 or 0 for motion and luminance features, 1 for texture features.
- BPN architecture: 6-3-1.
- BPN training: 500 epochs with 21 target region labels.
- $TH_{RS} = 50$.



(a) BPN output. (b) BPN output. (c) BPN output.

(d) final result. (e) final result. (f) final result.

(g) BPN output. (h) BPN output. (i) BPN output.

(j) final result. (k) final result. (l) final result.

Figure C.1: BPN-based segmentation results.

Appendix D

Segmentation Results Without Edge Fusion [2]

The parameters for the segmentation are as follows:

- Image size: gray-level QCIF.
- Local window size: 5×5.
- Feature Weighting: hard weighting,
  10 or 0 for motion and luminance features, 1 for texture features.
- SOFM architecture: 16 code words.
- SOFM training: 70,000 epochs.
- $TH_{eig} = 330$.
- $TH_{RS} = 10$.

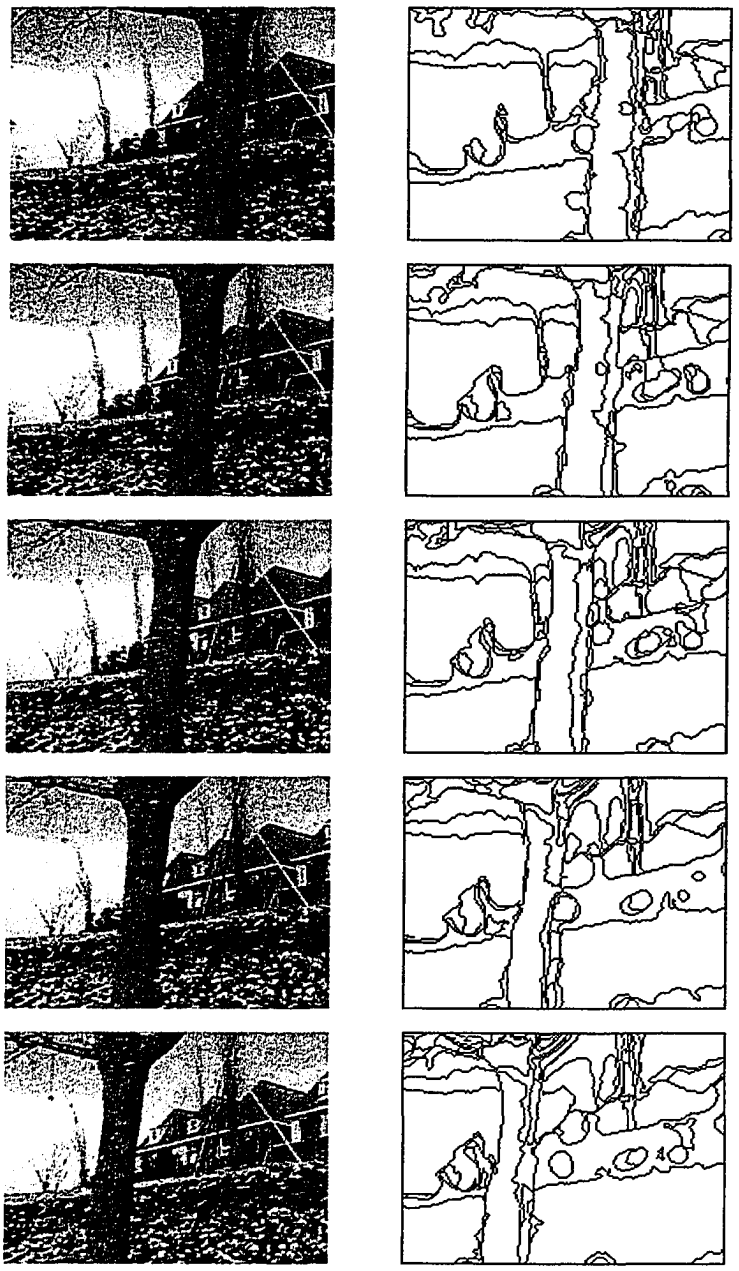


Figure D.1: Segmentation results without edge fusion (hard weighting) for *flower garden* sequence. First column: from the top, original frames 1, 5, 10, 15 and 20. Second column: from the top, segmentation results of frames 1, 5, 10, 15 and 20.

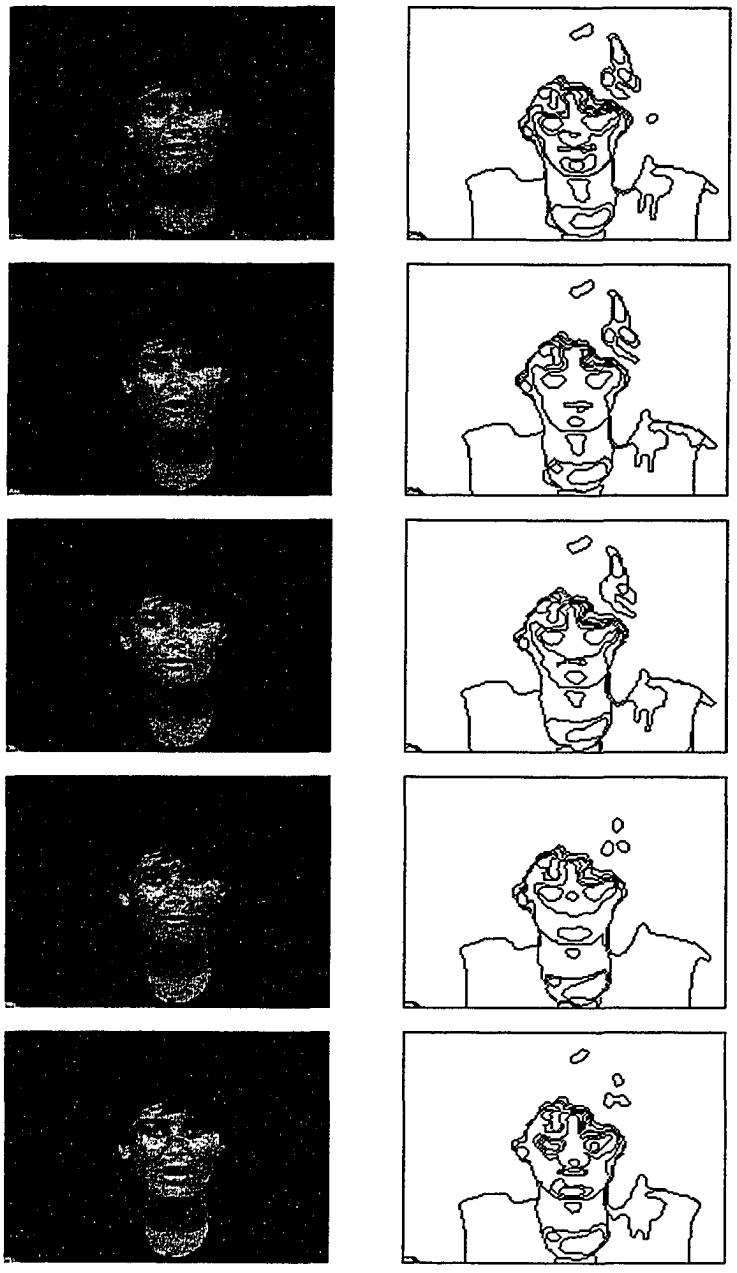


Figure D.2: Segmentation results without edge fusion (hard weighting) for *miss America* sequence. First column: from the top, original frames 1, 5, 10, 15 and 20. Second column: from the top, segmentation results of frames 1, 5, 10, 15 and 20.

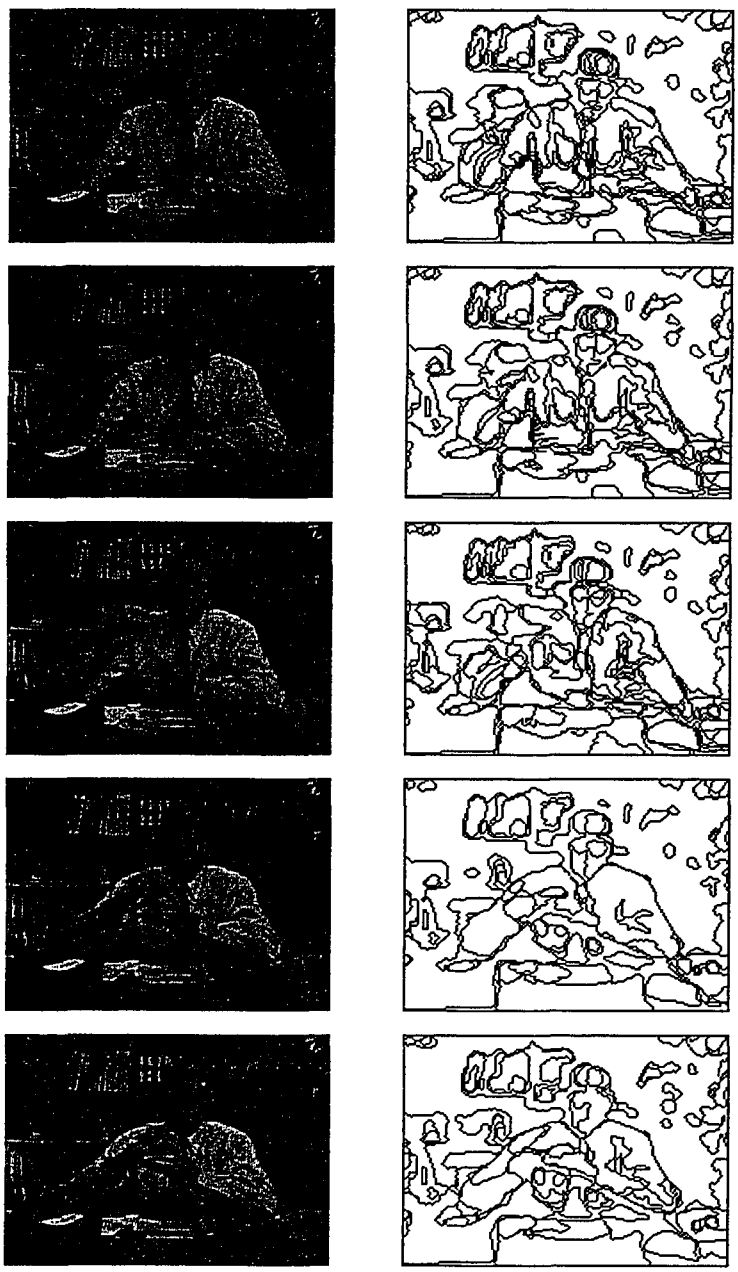


Figure D.3: Segmentation results without edge fusion (hard weighting) for *salesman* sequence. First column: from the top, original frames 1, 5, 10, 15 and 20. Second column: from the top, segmentation results of frames 1, 5, 10, 15 and 20.

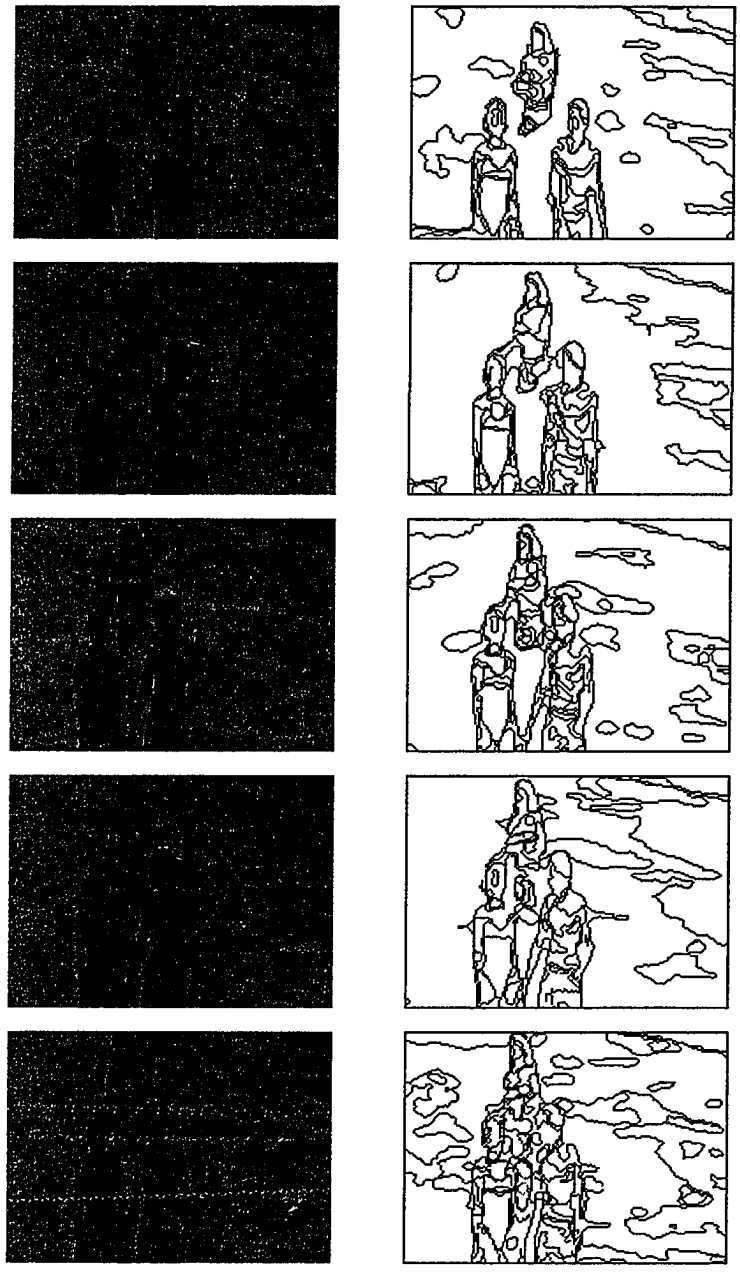


Figure D.4: Segmentation results without edge fusion (hard weighting) for *surf side* sequence. First column: from the top, original frames 1, 5, 10, 15 and 20. Second column: from the top, segmentation results of frames 1, 5, 10, 15 and 20.

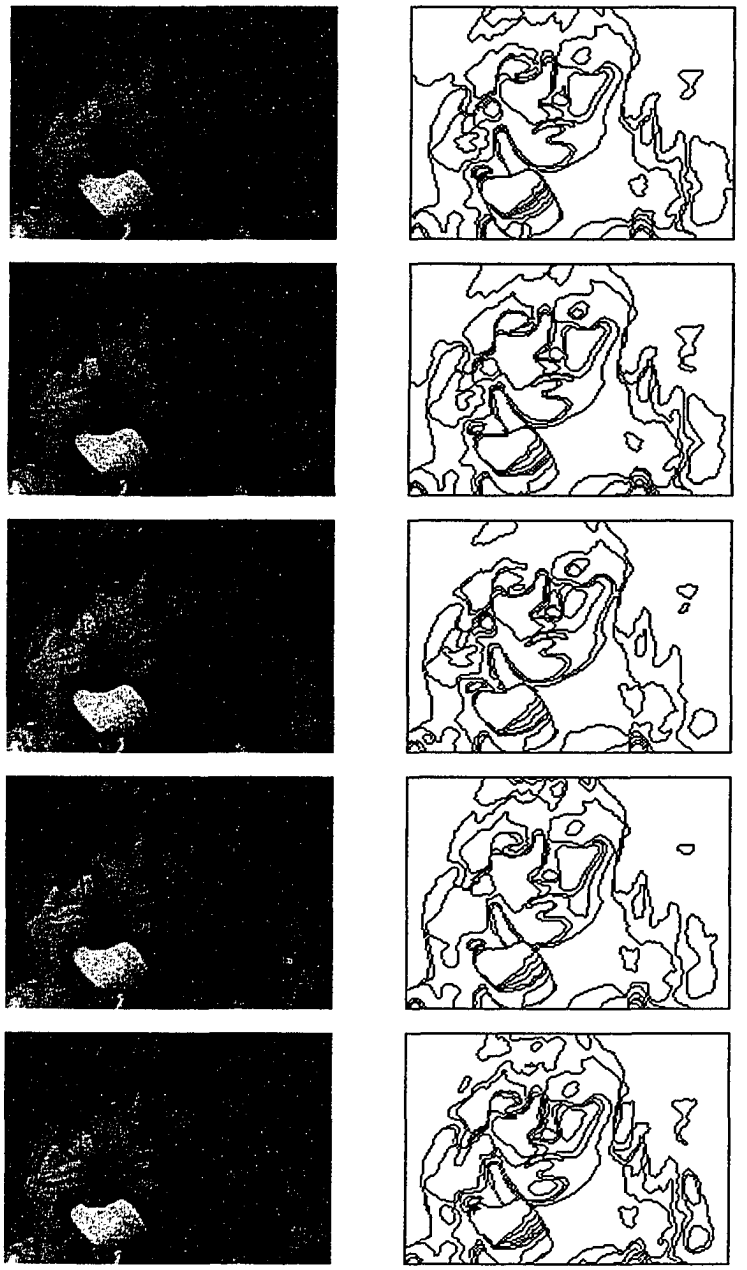


Figure D.5: Segmentation results without edge fusion (hard weighting) for *Susie* sequence. First column: from the top, original frames 1, 5, 10, 15 and 20. Second column: from the top, segmentation results of frames 1, 5, 10, 15 and 20.

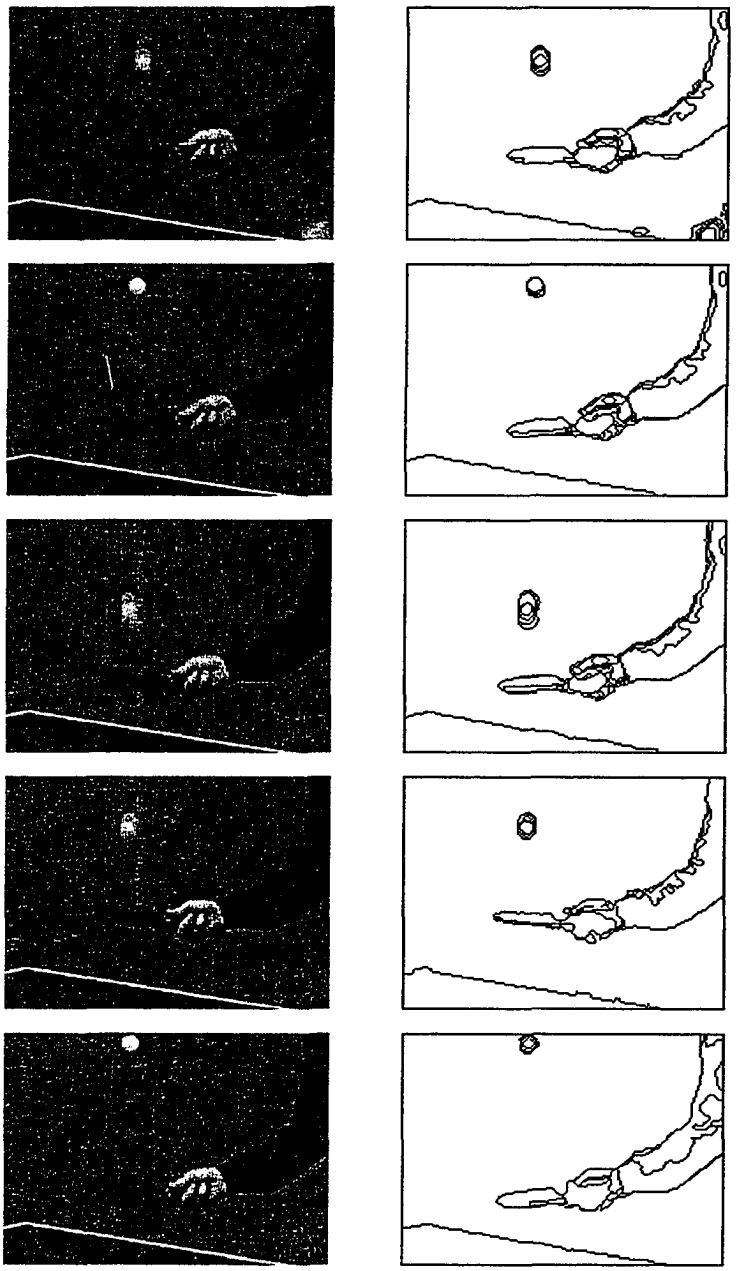


Figure D.6: Segmentation results without edge fusion (hard weighting) for *table tennis* sequence. First column: from the top, original frames 1, 5, 10, 15 and 20. Second column: from the top, segmentation results of frames 1, 5, 10, 15 and 20.

Appendix E

Detailed Evaluation Data for Number of Regions vs. Variance

Table E.1: Regions and Variance for *flower garden* image sequence (gray-level).

Proposed algorithm			Dorin's algorithm		
Parameter, α	No. of regions	Variance	Parameter, σ	No. of regions	Variance
5	318	11.61	2.9	790	133.64
10	142	18.58	3.0	749	144.35
20	66	73.23	3.5	216	253.85
30	49	82.18	4.0	210	244.95

Table E.2: Regions and Variance for *Miss America* image sequence (gray-level).

Proposed algorithm			Dorin's algorithm		
Parameter, α	No. of regions	Variance	Parameter, σ	No. of regions	Variance
5	368	15.86	2.9	217	29.06
10	185	36.61	3.0	155	33.20
20	79	80.73	3.5	113	39.65
30	50	196.89	4.0	69	67.56

Table E.3: Regions and Variance for *Salesman* image sequence (gray-level).

Proposed algorithm			Dorin's algorithm		
Parameter, α	No. of regions	Variance	Parameter, σ	No. of regions	Variance
5	659	34.17	2.9	642	56.82
10	397	46.66	3.0	643	56.86
20	211	96.02	3.5	396	83.52
30	142	177.50	4.0	383	82.20

Table E.4: Regions and Variance for *Susie* image sequence (gray-level).

Proposed algorithm			Dorin's algorithm		
Parameter, α	No. of regions	Variance	Parameter, σ	No. of regions	Variance
5	548	14.61	2.9	253	36.97
10	232	31.82	3.0	185	43.15
20	81	112.89	3.5	136	52.42
30	27	261.60	4.0	137	54.77

Table E.5: Regions and Variance for *Table tennis* image sequence (gray-level).

Proposed algorithm			Dorin's algorithm		
Parameter, α	No. of regions	Variance	Parameter, σ	No. of regions	Variance
5	137	17.54	2.9	495	49.91
10	109	18.00	3.0	267	62.03
20	73	23.27	3.5	192	83.60
30	52	25.75	4.0	186	81.03

Table E.6: Regions and Variance for *Carphone* image sequence (color).

Proposed algorithm			Dorin's algorithm		
Parameter, α	No. of regions	Variance	Parameter, σ	No. of regions	Variance
5	96	100.32	2.9	80	77.41
10	88	98.65	3.0	70	82.34
20	59	122.83	3.5	55	86.55
30	38	15.69	4.0	37	106.32

Table E.7: Regions and Variance for *Foreman* image sequence (color).

Proposed algorithm			Dorin's algorithm		
Parameter, α	No. of regions	Variance	Parameter, σ	No. of regions	Variance
5	73	47.51	2.9	81	58.71
10	65	53.76	3.0	78	59.59
20	47	66.07	3.5	38	80.86
30	29	92.98	4.0	37	92.54

Table E.8: Regions and Variance for *Miss America* image sequence (color).

Proposed algorithm			Dorin's algorithm		
Parameter, α	No. of regions	Variance	Parameter, σ	No. of regions	Variance
5	41	39.88	2.9	63	28.15
10	30	43.64	3.0	63	28.74
20	22	47.77	3.5	30	42.04
30	11	58.65	4.0	29	41.51

Table E.9: Regions and Variance for *Mother&daughter* image sequence (color).

Proposed algorithm			Dorin's algorithm		
Parameter, α	No. of regions	Variance	Parameter, σ	No. of regions	Variance
5	74	25.46	2.9	50	26.95
10	48	27.62	3.0	28	31.97
20	26	51.76	3.5	24	40.64
30	11	109.65	4.0	19	48.04

Table E.10: Regions and Variance for *Salesman* image sequence (color).

Proposed algorithm			Dorin's algorithm		
Parameter, α	No. of regions	Variance	Parameter, σ	No. of regions	Variance
5	138	49.78	2.9	149	62.12
10	100	54.96	3.0	115	72.82
20	55	67.40	3.5	66	94.89
30	33	111.40	4.0	50	105.03

Table E.11: Regions and Variance for *Table tennis* image sequence (color).

Proposed algorithm			Dorin's algorithm		
Parameter, α	No. of regions	Variance	Parameter, σ	No. of regions	Variance
5	29	54.27	2.9	22	71.81
10	19	53.32	3.0	26	68.01
20	17	53.38	3.5	13	76.92
30	16	53.33	4.0	7	131.07

Appendix F

Detailed Evaluation Data for Correct Segmentation Using Ground Truth

Table F.1: Correct segmentation for *Flow garden* image sequence (gray-level).

Tolerance %	GT Regions	Proposed		Dorins's	
		MS regions	Cor. regions	MS regions	Cor. regions
50	6	24	2	196	0
60	"	"	2	"	0
70	"	"	2	"	0
80	"	"	2	"	0
80	"	"	1	"	0

Table F.2: Correct segmentation for *Miss America* image sequence (gray-level).

Tolerance %	GT Regions	Proposed		Dorins's	
		MS regions	Cor. regions	MS regions	Cor. regions
50	29	73	10	73	7
60	"	"	9	"	6
70	"	"	8	"	5
80	"	"	5	"	4
80	"	"	4	"	4

Table F.3: Correct segmentation for *Salesman* image sequence (gray-level).

Tolerance %	GT Regions	Proposed		Dorins's	
		MS regions	Cor. regions	MS regions	Cor. regions
50	42	94	28	333	6
60	"	"	22	"	4
70	"	"	17	"	4
80	"	"	12	"	2
80	"	"	8	"	1

Table F.4: Correct segmentation for *Susie* image sequence (gray-level).

Tolerance %	GT Regions	Proposed		Dorins's	
		MS regions	Cor. regions	MS regions	Cor. regions
50	24	59	13	159	3
60	"	"	9	"	2
70	"	"	6	"	2
80	"	"	5	"	1
80	"	"	0	"	0

Table F.5: Correct segmentation for *Table tennis* image sequence (gray-level).

Tolerance %	GT Regions	Proposed		Dorins's	
		MS regions	Cor. regions	MS regions	Cor. regions
50	16	29	10	210	6
60	"	"	9	"	5
70	"	"	8	"	3
80	"	"	8	"	1
80	"	"	4	"	1

Table F.6: Correct segmentation for *Carphone* image sequence (color).

Tolerance %	GT Regions	Proposed		Dorins's	
		MS regions	Cor. regions	MS regions	Cor. regions
50	73	66	59	78	48
60	"	"	54	"	44
70	"	"	46	"	36
80	"	"	36	"	31
80	"	"	30	"	26

Table F.7: Correct segmentation for *Foreman* image sequence (color).

Tolerance %	GT Regions	Proposed		Dorins's	
		MS regions	Cor. regions	MS regions	Cor. regions
50	82	73	69	97	86
60	"	"	62	"	77
70	"	"	52	"	67
80	"	"	42	"	55
80	"	"	34	"	46

Table F.8: Correct segmentation for *Miss America* image sequence (color).

Tolerance %	GT Regions	Proposed		Dorins's	
		MS regions	Cor. regions	MS regions	Cor. regions
50	37	45	30	47	34
60	"	"	28	"	31
70	"	"	26	"	28
80	"	"	19	"	24
80	"	"	14	"	20

Table F.9: Correct segmentation for *Mother&daughter* image sequence (color).

Tolerance %	GT Regions	Proposed		Dorins's	
		MS regions	Cor. regions	MS regions	Cor. regions
50	37	50	29	36	25
60	"	"	26	"	23
70	"	"	24	"	20
80	"	"	17	"	18
80	"	"	9	"	14

Table F.10: Correct segmentation for *Salesman* image sequence (color).

Tolerance %	GT Regions	Proposed		Dorins's	
		MS regions	Cor. regions	MS regions	Cor. regions
50	44	57	37	86	30
60	"	"	30	"	24
70	"	"	27	"	20
80	"	"	21	"	18
80	"	"	16	"	13

Table F.11: Correct segmentation for *Table tennis* image sequence (color).

Tolerance %	GT Regions	Proposed		Dorins's	
		MS regions	Cor. regions	MS regions	Cor. regions
50	12	16	12	11	11
60	"	"	11	"	9
70	"	"	8	"	9
80	"	"	6	"	8
80	"	"	3	"	4

Appendix G

Number of Clock Cycles for the Segmentation Architecture

Note: The system parameters are the same as those used in Chapter 4. The unit of time is second.

Table G.1: Detailed time analysis of the VLSI architecture for QCIF image sequences.

Subsystems	Worst-case parameters		Mean parameters	
	No. of clocks	Time (400 MHz)	No. of clocks	Time (400 MHz)
Feat. ext.	6.653e5	16.632e-4	6.653e5	16.632e-4
Feat. filt.	1.127e5	3.172e-4	1.127e5	3.172e-4
Feat. norm.& weig.&labeling	0.255e5	0.636e-4	0.255e5	0.636e-4
Init. phase	0.818e5	20.440e-4	0.818e5	20.440e-4
Reg. Div.	1.269e5	3.172e-4	1.269e5	3.172e-4
Reg. Attr.	1.598e5	3.993e-4	1.115e5	2.788e-4
Neig. Info.	35.270e5	88.173e-4	19.032e5	47.580e-4
Energy cal.	24.389e5	60.971e-4	7.102e5	17.754e-4
Reg. Merg.	53.549e5	133.872e-4	5.899e5	14.743e-4
Combine	1.016e5	2.539e-4	1.016e5	2.539e-4
Sec. phase	117.087e5	292.717e-4	35.429e5	88.573e-4
Total	125.263e5	313.156e-4	43.605e5	109.012e-4

For the mean parameters, Avg. $N_{rl} = 430.6$, Avg. $RegL = 4.02$, $N_{loop} = 3.83$.

Table G.2: Detailed time analysis of the VLSI architecture for 256 × 256 image sequences.

Subsystems	Worst-case parameters		Mean parameters	
	No. of clocks	Time (400 MHz)	No. of clocks	Time (400 MHz)
Feat. ext.	17.204e5	43.001e-4	17.204e5	43.001e-4
Feat. filt.	3.279e5	8.196e-4	3.279e5	8.196e-4
Feat. norm.& weig.&labeling	0.656e5	1.640e-4	0.656e5	1.640e-4
Init. phase	21.138e5	52.844e-4	21.138e5	52.844e-4
Reg. Div.	3.280e5	8.199e-4	3.280e5	8.199e-4
Reg. Attr.	3.765e5	9.411e-4	3.007e5	7.518e-4
Neig. Info.	100.553e5	252.382e-4	55.845e5	139.612e-4
Energy cal.	48.119e5	120.296e-4	17.669e5	44.172e-4
Reg. Merg.	56.457e5	141.141e-4	27.658e5	69.144e-4
Combine	2.625e5	6.561e-4	2.625e5	6.561e-4
Sec. phase	214.795e5	526.986e-4	110.082e5	275.203e-4
Total	235.932e5	589.830e-4	131.219e5	328.047e-4

For the mean parameters, Avg. $N_{rl} = 648.6$, Avg. $RegL = 4.83$, $N_{loop} = 3.83$.

Table G.3: Detailed time analysis of the VLSI architecture for SIF image sequences.

Subsystems	Worst-case parameters		Mean parameters	
	No. of clocks	Time (400 MHz)	No. of clocks	Time (400 MHz)
Feat. ext.	22.546e5	56.364e-4	22.546e5	56.364e-4
Feat. filt.	4.296e5	10.740e-4	4.296e5	10.740e-4
Feat. norm.& weig.&labeling	0.860e5	2.149e-4	0.860e5	2.149e-4
Init. phase	27.702e5	69.253e-4	27.702e5	69.253e-4
Reg. Div.	4.297e5	10.743e-4	4.297e5	10.743e-4
Reg. Attr.	4.625e5	12.000e-4	4.483e5	11.000e-4
Neig. Info.	136.398e5	341.000e-4	89.350e5	223.000e-4
Energy cal.	24.389e5	61.000e-4	9.271e5	23.000e-4
Reg. Merg.	313.711e5	784.000e-4	44.206e5	111.000e-4
Combine	3.483e5	8.595e-4	3.438e5	8.595e-4
Sec. phase	486.860e5	1217.000e-4	155.040e5	388.000e-4
Total	514.560e5	1286.000e-4	182.750e5	457.000e-4

For the mean parameters, Avg. $N_{rl} = 430.6$, Avg. $RegL = 13.63$, $N_{loop} = 3.83$.

Table G.4: Detailed time analysis of the VLSI architecture for CCIR601 image sequences.

Subsystems	Worst-case parameters		Mean parameters	
	No. of clocks	Time (400 MHz)	No. of clocks	Time (400 MHz)
Feat. ext.	90.720e5	226.800e-4	90.720e5	226.800e-4
Feat. filt.	17.282e5	43.204e-4	17.282e5	43.204e-4
Feat. norm.& weig.&labeling	3.547e5	8.642e-4	3.547e5	8.642e-4
Init. phase	111.459e5	278.646e-4	111.459e5	278.646e-4
Reg. Div.	17.285e5	43.213e-4	17.285e5	43.213e-4
Reg. Attr.	17.610e5	44.000e-4	17.468e5	44.000e-4
Neig. Info.	871.674e5	2179.000e-4	599.439e5	1499.000e-4
Energy cal.	24.388e5	61.000e-4	9.271.000e5	23.000e-4
Reg. Merg.	2444.808e5	6112.000e-4	249.930e5	875.000e-4
Combine	13.829e5	34.573e-4	13.829e5	34.573e-4
Sec. phase	3389.600e5	8474.000e-4	1007.200e5	2518.000e-4
Total	3501.100e5	8753.000e-4	1118.700e5	2729.000e-4

For the mean parameters, Avg. $N_{rl} = 430.6$, Avg. $RegL = 54.82$, $N_{loop} = 3.83$.

Appendix H

Hardware Complexity for the Segmentation Architecture

Note: Image size temporary buffers for each subsystem marked as "-", are shared. Please see the memory requirements for entire system. The system parameters are the same as those used in Chapter 4.

Table H.1: Hardware complexity of the VLSI architecture for the QCIF image sequences.

Sub-systems	Memory (bit)	Registers (avg. width)	Adders (avg. width)	Multipliers (avg. width)	Comparators (avg. width)
Feature extraction	-	1993 (9.54)	232 (11.56)	27 (8.0)	2 (10.50)
Feature filtering	-	2272 (1)	0 (0)	0 (0)	0 (0)
Feature norm.&weig.	-	66 (8.85)	7 (8.0)	21 (8.0)	0 (0)
Feature labeling	-	1883 (11.80)	392 (33.00)	343 (16.00)	49 (35.00)
Initial phase	-	6154 (7.08)	631 (24.84)	391 (15.02)	51 (34.04)
Region division	0.025 M	655 (5.99)	0 (0)	0 (0)	5 (8.0)
EF: region attributes	0.38 M	50 (17.48)	4 (8.0)	0 (0)	14 (13.72)
EF: neigh. info.	0.05 M	308 (5.83)	2 (6.0)	0 (0)	5 (6.0)
EF: energy calculation	0 M	37 (11.25)	8 (8.0)	7 (23.00)	3 (10.34)
EF: region merging	0 M	12 (17.50)	4 (15.00)	0 (0)	2 (10.50)
EF: total	0.43 M	407 (8.10)	18 (9.34)	7 (23.00)	24 (11.42)
Text: region attributes	0.25 M	12 (8.50)	3 (8.0)	0 (0)	2 (6.0)
Text: neigh. info.	0.001 M	291 (6.01)	0 (0)	0 (0)	5 (6.0)
Text: energy calculation	0 M	12 (14.00)	4 (8.0)	4 (23.00)	2 (10.50)
Text: region merging	0 M	10 (16.40)	3 (15.00)	0 (0)	2 (10.50)
Text: total	0.26 M	325 (6.71)	10 (10.10)	4 (23.00)	11 (7.82)
Combine	-	580 (6.00)	0 (0)	0 (0)	8 (8.0)
Second phase	0.71 M	1967 (6.55)	28 (9.61)	11 (23.00)	48 (9.67)
Entire system	3.76 M	82131 (6.95)	659 (24.19)	402 (15.24)	99 (22.23)

Table H.2: Hardware complexity of the VLSI architecture for the 256 × 256 image sequences.

Sub-systems	Memory (bit)	Registers (avg. width)	Adders (avg. width)	Multipliers (avg. width)	Comparators (avg. width)
Feature extraction	-	1993 (9.54)	232 (11.56)	27 (8.0)	2 (10.50)
Feature filtering	-	2272 (1)	0 (0)	0 (0)	0 (0)
Feature norm.&weig.	-	66 (8.85)	7 (8.0)	21 (8.0)	0 (0)
Feature labeling	-	1883 (11.80)	392 (33.00)	343 (16.00)	49 (35.00)
Initial phase	-	6154 (7.21)	631 (24.84)	391 (15.02)	51 (34.04)
Region division	0.066 M	1159 (5.99)	0 (0)	0 (0)	5 (8.0)
EF: region attributes	0.59 M	50 (18.24)	4 (8.0)	0 (0)	14 (14.58)
EF: neigh. info.	0.077 M	532 (5.91)	2 (6.5)	0 (0)	5 (6.0)
EF: energy calculation	0 M	37 (11.57)	8 (8.0)	7 (24.00)	3 (10.67)
EF: region merging	0 M	12 (18.34)	4 (16.00)	0 (0)	2 (11.00)
EF: total	0.66 M	631 (7.45)	18 (9.62)	7 (24.00)	24 (12.00)
Text: region attributes	0.39 M	12 (8.66)	3 (8.0)	0 (0)	2 (6.0)
Text: neigh. info.	0.003 M	515 (6.01)	0 (0)	0 (0)	5 (6.0)
Text: energy calculation	0 M	12 (14.67)	4 (8.0)	4 (24.00)	2 (12.00)
Text: region merging	0 M	10 (17.20)	3 (16.00)	0 (0)	2 (11.00)
Text: total	0.40 M	549 (6.46)	10 (10.40)	4 (24.00)	11 (8.00)
Combine	-	1028 (6.00)	0 (0)	0 (0)	8 (8.0)
Second phase	1.13 M	3367 (6.35)	28 (9.99)	11 (24.00)	48 (10.00)
Entire system	8.99 M	10529 (6.93)	659 (24.21)	402 (15.27)	99 (22.29)

Table H.3: Hardware complexity of the VLSI architecture for the SIF image sequences.

Sub-systems	Memory (bit)	Registers (avg. width)	Adders (avg. width)	Multipliers (avg. width)	Comparators (avg. width)
Feature extraction	-	2833 (9.01)	232 (11.56)	27 (8.0)	2 (10.50)
Feature filtering	-	2272 (1)	0 (0)	0 (0)	0 (0)
Feature norm.&weig.	-	66 (8.85)	7 (8.0)	21 (8.0)	0 (0)
Feature labeling	-	1883 (11.80)	392 (33.00)	343 (16.00)	49 (35.00)
Initial phase	-	7054 (7.19)	631 (24.84)	391 (15.02)	51 (34.04)
Region division	0.086 M	1105 (6.60)	0 (0)	0 (0)	5 (8.0)
EF: region attributes	1.42 M	50 (19.43)	4 (8.0)	0 (0)	14 (15.63)
EF: neigh. info.	0.090 M	508 (5.90)	2 (6.5)	0 (0)	5 (6.0)
EF: energy calculation	0 M	37 (11.84)	8 (8.0)	7 (25.00)	3 (11.00)
EF: region merging	0 M	12 (19.17)	4 (17.00)	0 (0)	2 (11.50)
EF: total	1.51 M	631 (7.76)	18 (9.84)	7 (25.00)	26 (12.93)
Text: region attributes	0.97 M	12 (8.84)	3 (8.0)	0 (0)	2 (6.0)
Text: neigh. info.	0.003 M	491 (6.01)	0 (0)	0 (0)	5 (6.0)
Text: energy calculation	0 M	12 (15.34)	4 (8.0)	4 (25.00)	2 (12.50)
Text: region merging	0 M	10 (18.00)	3 (17.00)	0 (0)	2 (11.50)
Text: total	0.97 M	525 (6.51)	10 (10.70)	4 (25.00)	11 (8.19)
Combine	-	980 (6.00)	0 (0)	0 (0)	8 (8.0)
Second phase	2.56 M	3223 (6.42)	28 (10.15)	11 (25.00)	50 (10.60)
Entire system	12.87 M	10277 (6.95)	659 (24.22)	402 (15.30)	101 (22.44)

Table H.4: Hardware complexity of the VLSI architecture for the CCIR 601 image sequences.

Sub-systems	Memory (bit)	Registers (avg. width)	Adders (avg. width)	Multipliers (avg. width)	Comparators (avg. width)
Feature extraction	-	4957 (8.60)	232 (11.56)	27 (8.0)	2 (10.50)
Feature filtering	-	2272 (1)	0 (0)	0 (0)	0 (0)
Feature norm.&weig.	-	66 (8.85)	7 (8.0)	21 (8.0)	0 (0)
Feature labeling	-	1883 (11.80)	392 (33.00)	343 (16.00)	49 (35.00)
Initial phase	-	9178 (6.93)	631 (24.24)	391 (15.35)	51 (22.72)
Region division	0.35 M	2167 (6.00)	0 (0)	0 (0)	5 (8.0)
EF: region attributes	6.24 M	62 (21.39)	4 (8.0)	0 (0)	18 (17.56)
EF: neigh. info.	0.20 M	980 (5.95)	2 (7.0)	0 (0)	5 (6.0)
EF: energy calculation	0 M	37 (12.44)	8 (8.0)	7 (27.00)	3 (11.67)
EF: region merging	0 M	12 (20.84)	4 (19.00)	0 (0)	2 (12.50)
EF: total	6.44 M	1091 (7.21)	18 (10.34)	7 (27.00)	28 (14.50)
Text: region attributes	4.25 M	12 (9.17)	3 (8.0)	0 (0)	2 (6.0)
Text: neigh. info.	0.008 M	963 (6.01)	0 (0)	0 (0)	5 (6.0)
Text: energy calculation	0 M	12 (16.67)	4 (8.0)	4 (27.00)	2 (13.50)
Text: region merging	0 M	10 (19.60)	3 (19.00)	0 (0)	2 (12.50)
Text: total	4.26 M	997 (6.31)	10 (11.30)	4 (27.00)	11 (8.56)
Combine	-	1924 (6.00)	0 (0)	0 (0)	8 (8.0)
Second phase	11.04 M	6179 (6.27)	28 (10.68)	11 (27.00)	52 (11.62)
Entire system	52.52 M	15357 (6.93)	659 (24.24)	402 (15.35)	103 (22.72)

Appendix I

Source Codes for the Segmentation Algorithm

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [colFeat] = colFeatGen(fn, dim, ws, disp)
%colFeatGen  feature vector genreration  for color image(YUV format).
%   Motion vectors and U and V components are scaled up to 128.
%  Also, Eig values are truncated to unit8 because the higer values
%  give the same effect as original in our implementation.
%  ALL THE FEATURE ARE 1 Byte RANGES !!!.
%
% RETURNS
%  ComFeat   Combined feature vectors. [Y U V motX motY STD Ymean EIG] for each pixel.
%            data are stored as colummwise.
%            .. 1st col -> 2nd col. of input image 2D.
%
% Copyright (c) 1999 by JinSang Kim, CSU EE.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

error(nargchk(4, 4, nargin))  % check no. of input args is correct
fnCol = fn.col; fnMotX = fn.motX; fnMotY = fn.motY; fnPS = fn.ps;
total = dim(1,1)*dim(1,2);

% read a color image and convert to YUV format.
colImg = imread(fnCol,'bmp'); yuv = rgb2yuv(colImg); intImg=yuv(:,:,1);

% Motion, esp. for Anandan displacement vectors.
fidMotX = fopen(fnMotX,'r');
if fidMotX < 0
 error(['Cannot open ' fnMotX]);
end
MotX = fread(fidMotX,'uchar');  %Column vector.
% replace NaN value to '0'
MotX = nanops(MotX);

MotX = reshape(MotX,dim(1,1),dim(1,2)); % not transposed

% MotY
fidMotY = fopen(fnMotY,'r');
if fidMotY < 0
 error(['Cannot open ' fnMotY]);
end
 MotY = fread(fidMotY,'uchar');
% replace NaN value to '0'
MotY = nanops(MotY);
```

```
 MotY = reshape(MotY,dim(1,1),dim(1,2)); % not transposed
 fclose('all');

% Texture from an intensity Image( std and Imean).
 mws = 9; order = 10; Imean = colfilt(intImg,[ws ws],'sliding','mean');
 Istd = colfilt(double(intImg),[ws ws],'sliding','std');
 Istd = colfilt(Istd,[mws mws],'sliding','minfilt',order);

% texturedness measure(confidence value)(eigenvalue calculattion).
% min filtering
[gxo,gyo] = gradient(double(intImg));
 eigVal = smEigen(gxo,gyo,ws); % choose small eigenvalue.
 minws = 5; order = 7;
 Eig = colfilt(eigVal,[minws minws],'sliding','minfilt',order); % min filtering.

 % median filtering if necessary.
 'Now median filtering ....' t = cputime; Y =
 colfilt(double(intImg),[ws ws],'sliding','median'); U =
 colfilt(double(yuv(:,:,2)),[ws ws],'sliding','median'); V =
 colfilt(double(yuv(:,:,3)),[ws ws],'sliding','median'); U = U +
 128; V = V + 128;

 MotX = colfilt(MotX,[ws ws],'sliding','median');
 MotY = colfilt(MotY,[ws ws],'sliding','median');
 Istd = colfilt(Istd,[ws ws],'sliding','median');
 Imean = colfilt(Imean,[ws ws],'sliding','median');
 Eig =  colfilt(Eig,[ws ws],'sliding','median');
 if(disp == 1)
   figure; orient tall;
   subplot(421); imshow(colImg);title('color image');
   subplot(422); imshow(uint8(yuv(:,:,1)));title('Y');
   subplot(423); imshow(uint8(U));title('U(scaled to display)');
   subplot(424); imshow(uint8(V));title('V(scaled to display)');
   subplot(425); imshow(uint8(abs(MotX+MotY.*i)));title('Mot magnitude');
   subplot(426); imshow(mat2gray(Istd));title('STD');
   subplot(427); imshow(uint8(Imean));title('Y mean');
   subplot(428); imshow(uint8(Eig));title('EIG:truncated, 0 ~ 255');
   feval('print','-dps',fnPS);
 end

 % reshaping.
 Y = reshape(yuv(:,:,1),total,1); U = reshape(U,total,1); V =
 reshape(V,total,1); MotX = reshape(MotX,total,1); MotY =
 reshape(MotY,total,1); Istd = reshape(Istd,total,1); Imean =
 reshape(Imean,total,1); Eig = reshape(Eig,total,1);

 'Now Combining feature data into one file'
 colFeat = [Y U V MotX MotY Istd Imean Eig ];
 colFeat = uint8(colFeat);

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 function [rx] = smEigen(gx,gy,ws)
 % [rx] =  smEigen(gx,gy,ws)
 %         smaller eigenvalue of Z matrix for Optical
 %         flow contraint equation to measure texturedness of
 %         input image x.
 %  By JinSang Kim CSU ECE 1999.

 [r,c] = size(gx);
 % weighting window for give more importance at center of window.
 % gaussian - like window for ONLY 5x5 window.
 if(ws == 5)
 ww = [ .125, .25, .25, .25, .125;
        .25 , .50, .75, .50, .25;
```

```
        .25 , .75,   1, .75, .25;
        .25 , .50, .75, .50, .25;
        .125, .25, .25, .25, .125];
elseif(ws == 3)
ww = [.50, .75, .50, ;
      .75,   1, .75, ;
      .50, .75, .50, ];
else
 ww = ones(ws,ws);
end

ww = reshape(ww,ws*ws,1);
gx2 = colfilt(gx,[ws ws],'sliding','gsqsum',ww);
gy2 = colfilt(gy,[ws ws],'sliding','gsqsum',ww);
tmp = gx .* gy; gxy = colfilt(tmp,[ws ws],'sliding','gsum',ww);

% gradient values should be positive.
gx2 = reshape(gx2,r*c,1);
gy2 = reshape(gy2,r*c,1);
gxy = abs(reshape(gxy,r*c,1));

% smaller eigenvalue of Z matrix.
zmat =[gx2';gxy';gxy';gy2']; % 4 x r*c matrix
zmat = col2im(zmat,[2 2],[2 2*r*c],'distict'); rx =
blkproc(zmat,[2 2],'smallEigVal'); rx = reshape(rx,r,c);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function net = somCol(Epochs, codeSize, Comment,normFlag, params,...
                      THconf,savePath,SUBSAMPLE)
%  Training SOFM  with soft weighting.
%
%  Epochs       epochs for training
%  Codesize     codebook size ex: [ 4 8 ];
%  Commnet      comment for title, should be string array. ex: '10K_16x16';
%                 should have blank like this : " test"
% RETURN
%  net          (struct) network information.
%  By JinSang Kim CSU ECE 1999.

 params = params(:,(1:7)); %exclude the confidence measure.
 normFunc = 'normal_ts';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Load input data and ramdom sampling
% according to given featureIndex
% data are subsampled because of "OUT OF MEMORY" problems.
%
 dim = [288 360];
 load claire0Feat.mat; % calt0_256 loaded.
   EIG = double(colFeat(:,8)); %confidence value, Eigenvalue.
   colFeat = double(colFeat(:,(1:7)));
   colFeat = feval(normFunc,colFeat,params);
   colFeat = adaptWtColKIM(colFeat,EIG,THconf);
   I1 = colFeat(1:SUBSAMPLE:dim(1,1)*dim(1,2),:);

 dim = [240 352];
 load football0Feat.mat;
   EIG = double(colFeat(:,8)); %confidence value, Eigenvalue.
   colFeat = double(colFeat(:,(1:7)));
   colFeat = feval(normFunc,colFeat,params);
   colFeat = adaptWtColKIM(colFeat,EIG,THconf);
   I2 = colFeat(1:SUBSAMPLE:dim(1,1)*dim(1,2),:);

 dim = [240 352];
```

```
 load garden0Feat.mat;
   EIG = double(colFeat(:,8)); %confidence value, Eigenvalue.
   colFeat = double(colFeat(:,(1:7)));
   colFeat = feval(normFunc,colFeat,params);
   colFeat = adaptWtColKIM(colFeat,EIG,THconf);
   I3 = colFeat(1:SUBSAMPLE:dim(1,1)*dim(1,2),:);

 dim = [512 512];
 load mall0Feat.mat;
   EIG = double(colFeat(:,8)); %confidence value, Eigenvalue.
   colFeat = double(colFeat(:,(1:7)));
   colFeat = feval(normFunc,colFeat,params);
   colFeat = adaptWtColKIM(colFeat,EIG,THconf);
   I4 = colFeat(1:SUBSAMPLE:dim(1,1)*dim(1,2),:);

 dim = [512 512];
 load mit0Feat.mat;
   EIG = double(colFeat(:,8)); %confidence value, Eigenvalue.
   colFeat = double(colFeat(:,(1:7)));
   colFeat = feval(normFunc,colFeat,params);
   colFeat = adaptWtColKIM(colFeat,EIG,THconf);
   I5 = colFeat(1:SUBSAMPLE:dim(1,1)*dim(1,2),:);

 dim = [240 352];
 load mobile0Feat.mat;
   EIG = double(colFeat(:,8)); %confidence value, Eigenvalue.
   colFeat = double(colFeat(:,(1:7)));
   colFeat = feval(normFunc,colFeat,params);
   colFeat = adaptWtColKIM(colFeat,EIG,THconf);
   I6 = colFeat(1:SUBSAMPLE:dim(1,1)*dim(1,2),:);

 TSample =  [I1;I2;I3;I4;I5;I6]; % normalized feature vector.
 clear I* colFeat EIG;

 [MinData,Yi] = min(TSample);
 [MaxData,Ya] = max(TSample);

%---------------
% Training SOMF
%---------------
 MinMaxData = [MinData;MaxData];
 net = newsom(MinMaxData',codeSize);
 net.trainParam.epochs = Epochs;
 net = train(net,TSample');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function somColTest(fnnet, codeSize,Comment,normFlag,params,THconf,savePath)
%  SOFM  testing with adaptive weighting.
%
%  Epochs       epochs for training
%  Codesize     codebook size ex: [ 7 7 ];
%  Commnet      comment for title, should be string array. ex: '10K_16x16';
%                 should have blank like this : " test"
%
%  By JinSang Kim CSU ECE 1999.

params = params(:,(1:7)); %exclude the confidence measure.
normFunc = 'normal_ts';

% load trained network.
feval('load',fnnet);%net codeSize Epochs are loaded.

%-------------------------
% Testing SOFM with WEIGHT
```

```
%-------------------------
NCOD = codeSize(1,1)*codeSize(1,2);
cwd = pwd;
dim = [ 144 176 ]; % QCIF image.
feval('chdir',savePath);

figure
orient tall;

load carphone0Feat.mat; % calt0_256 loaded.
subplot(321)
 EIG = double(colFeat(:,8)); %confidence value, Eigenvalue.
 colFeat = double(colFeat(:,(1:7)));
 colFeat = feval(normFunc,colFeat,params);
 colFeat = adaptWtColKIM(colFeat,EIG,THconf);
 Labl = sim(net,(colFeat)');
 SofmOut = somGetLabels1(Labl,codeSize,dim);
 colored = ind2rgb(SofmOut,[0 0 0 ; jet(max(NCOD))]); imshow(colored);
 tname = strcat('carphone0',Comment,normFlag);
 title(tname);
 feval('save',tname,'SofmOut');

load foreman0Feat.mat; subplot(322)
 EIG = double(colFeat(:,8)); %confidence value, Eigenvalue.
 colFeat = double(colFeat(:,(1:7)));
 colFeat = feval(normFunc,colFeat,params);
 colFeat = adaptWtColKIM(colFeat,EIG,THconf);
 Labl = sim(net,(colFeat)');
 SofmOut = somGetLabels1(Labl,codeSize,dim);
 colored = ind2rgb(SofmOut,[0 0 0 ; jet(max(NCOD))]); imshow(colored);
 tname = strcat('foreman0',Comment,normFlag);
 title(tname);
 feval('save',tname,'SofmOut');

load missAm0Feat.mat; subplot(323)
 EIG = double(colFeat(:,8)); %confidence value, Eigenvalue.
 colFeat = double(colFeat(:,(1:7)));
 colFeat = feval(normFunc,colFeat,params);
 colFeat = adaptWtColKIM(colFeat,EIG,THconf);
 Labl = sim(net,(colFeat)');
 SofmOut = somGetLabels1(Labl,codeSize,dim);
 colored = ind2rgb(SofmOut,[0 0 0 ; jet(max(NCOD))]); imshow(colored);
 tname = strcat('missAm0',Comment,normFlag);
 title(tname);
 feval('save',tname,'SofmOut');

load mom&dau0Feat.mat; subplot(324)
 EIG = double(colFeat(:,8)); %confidence value, Eigenvalue.
 colFeat = double(colFeat(:,(1:7)));
 colFeat = feval(normFunc,colFeat,params);
 colFeat = adaptWtColKIM(colFeat,EIG,THconf);
 Labl = sim(net,(colFeat)');
 SofmOut = somGetLabels1(Labl,codeSize,dim);
 %SofmOut = mat2gray(SofmOut,[0 NCOD]);
 colored = ind2rgb(SofmOut,[0 0 0 ; jet(max(NCOD))]); imshow(colored);
 tname = strcat('mom&dau0',Comment,normFlag);
 title(tname);
 feval('save',tname,'SofmOut');

load salesman0Feat.mat;
subplot(325)
 EIG = double(colFeat(:,8)); %confidence value, Eigenvalue.
 colFeat = double(colFeat(:,(1:7)));
 colFeat = feval(normFunc,colFeat,params);
```

```
 colFeat = adaptWtColKIM(colFeat,EIG,THconf);
 Labl = sim(net,(colFeat)');
 SofmOut = somGetLabels1(Labl,codeSize,dim);
 colored = ind2rgb(SofmOut,[0 0 0 ; jet(max(NCOD))]); imshow(colored);
 tname = strcat('salesman0',Comment,normFlag);
 title(tname);
 feval('save',tname,'SofmOut');


load table0Feat.mat;
subplot(326)
 EIG = double(colFeat(:,8)); %confidence value, Eigenvalue.
 colFeat = double(colFeat(:,(1:7)));
 colFeat = feval(normFunc,colFeat,params);
 colFeat = adaptWtColKIM(colFeat,EIG,THconf);
 Labl = sim(net,(colFeat)');
 SofmOut = somGetLabels1(Labl,codeSize,dim);
 colored = ind2rgb(SofmOut,[0 0 0 ; jet(max(NCOD))]); imshow(colored);
 tname = strcat('table0',Comment,normFlag);
 title(tname);
 feval('save',tname,'SofmOut');

 str = strcat('col',Comment,normFlag);
 feval('print','-dpsc',str);
 feval('chdir',cwd);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 function colEdgeFusion(THsmall,THstat,marg,THeig,conComp,...
                        THedge,func,savePath,fn,colFv,dim,comment)
%  Edge fusion following an Algorithm.
%     THsmall : small region removal.
%     THstat,THedge: parameters for edge fusion.
%     marg: image boundary margin to be deleted.
%     comComp: 4 or 8 connected component analysis.
%  By JinSang Kim CSU ECE 1999.

% show NN output
 feval('load',fn.nnout);% SofmOut or SgmtImg loaded.

 if(findstr(comment,'P'))  % for loading postprocessing output.
    postMarg = marg-5;
    SofmOut = remBndaryOne(SgmtImg,postMarg);
    [rtmp,ctmp] = size(SgmtImg);
    tmp = ones(rtmp+2*5,ctmp+2*5); % for colFv.
    [dummy,colFv] = remBndary(tmp,colFv,marg);
 else
    [SofmOut,colFv] = remBndary(SofmOut,colFv,marg);
    tmp = SofmOut;
 end

 %[SofmOut,colFv] = remBndary(SofmOut,colFv,postMarg);
 [r,c] = size(SofmOut);
 coloredOrg = ind2rgb(SofmOut,[0 0 0 ; jet(max(SofmOut(:)))]);
 Nreg = getLabelFromRegion(SofmOut,conComp);

 'MAX number of regions of NN output'
 MAXR = max(Nreg(:))
 % get the labeled region statics for NN outputs. (region length).
 % used for HW region access time.
 [AvgRegLenth, maxLeng, minLeng, NR] = regLengthCal(Nreg,conComp)

  if(~isempty(fn.smNNout))
     feval('load',fn.smNNout); LblReg = getLabelFromRegion(LblReg,conComp);
  else
```

```
    'small region remval start'
    LblReg = colSmallRegMerg(colFv,SofmOut,THsmall,THstat,conComp);
    feval('save',fn.smNNout,'LblReg');
    'small region remval end'
 end
imshow(mat2gray(LblReg));
smNNout = LblReg;
max(smNNout)
NNcont = plotCont(smNNout);
imwrite(uint8(255*NNcont),gmap,'carphoneNC.bmp','bmp');
figure;imshow(NNcont);

coloredOrg = ind2rgb(smNNout,[0 0 0 ; jet(max(smNNout(:)))]);
smNNout = getLabelFromRegion(smNNout,conComp);

% mask generation for texture regions.
% motion confidence data, 'filtEIG' loaded.
filtEIG = colFv(:,8); % eigenvalues.

maskDim =[dim(1,1)-2*marg dim(1,2)-2*marg]; maskMarg = 0;
textMask = colTextMaskGenOnlyConf(THeig,THsmall,filtEIG, maskDim, maskMarg);

% merge texture regions with 'TEXTURE DISTANCE';
Dist = 'NoWGT'; % assume no weights.
figure; imshow(textMask);
[textMergReg] = colTextRegOrderMergMask(smNNout,colFv,...
                THsmall,THstat,Dist,textMask,conComp,comment);
[textMergReg] = colSmallRegMerg(colFv,textMergReg,THsmall,...
                 THstat,conComp);  %small regions after splitting.
figure; imshow(mat2gray(textMergReg));

% non texture regions and determine the perimeter(contours).
nonTextRegR = zeros(r*c,1); % 0 is background.
[i,dummy] = find(reshape(textMask,r*c,1) == 0); % non texture regions.
smNNoutR = reshape(smNNout,r*c,1);
nonTextRegR(i,1) = smNNoutR(i,1);
nonTextReg = reshape(nonTextRegR,r,c);
orgNR = max(nonTextReg(:));

% load edge linked map for orignal images. value 0's are edge.
[edge,map] = feval('imread',fn.lowEdge,'bmp');
[edge] = remBndaryOne(edge,marg);
edge = (double(edge) / 255); max(edge(:));
% overwrite the edge over nontexture regions and split the region broken by edges.
maxNTR = max(nonTextReg(:));

coloredOrg = ind2rgb(smNNout,[0 0 0 ; jet(max(smNNout(:)))]);
str = strcat('# of max Regs =',num2str(max(nonTextReg(:)))); ylabel(str);

edgeDisp = 128 *abs(double(edge)-1);
[eoi,dummy] = find(reshape(edgeDisp,r*c,1));

edgeNTextReg = nonTextRegR; edgeNTextReg(eoi,:) = 128;
edgeNTextReg = reshape(edgeNTextReg,r,c);

[NTELbl]= regSplitEdgeEZero(edge,conComp,nonTextReg);

edgeSR = max(NTELbl(:));
coloredEdge = ind2rgb(NTELbl,[0 0 0 ; jet(max(NTELbl(:)))]);

% save regions splitted by edges.
curPath = pwd;

% merging nonTexture regions.
```

```
% Assume  the Mask is ACCURATE after refining by using edge info.
% and an idea how to merge regions between texture and nontexture bondaries.
%
NTELbl = getLabelFromRegionEZero(NTELbl,conComp);
if(~isempty(fn.highEdge))
    [highEdge,map] = feval('imread',fn.highEdge,'bmp');
    %edge = uint8(double(edge)/255); % to binary edge map.
      [highEdge] = remBndaryOne(highEdge,marg);
      highEdge = (double(highEdge) / 255); max(highEdge(:));
else
     highEdge = edge;
end

stat = struct('orgR',orgNR,'edgeSR',edgeSR,'wgcPer',[],'wbndPer',[],'edgeMergR',[]);

% do Edge Fusion on nonTexture regions.
% this function is exactly same as gray one except feature(colFV).
[mergELbl, stat]= colMergWithEdge(edge,NTELbl,highEdge,colFv(:,1:3),...
                              THedge,func,conComp,stat,THsmall);

% merge two mask regions.
[mergLbl] = merg2MaskReg(mergELbl,textMergReg,textMask,conComp);

% show merged regions.
maxR = max(mergLbl(:));
disp(['# of regions after merging with edges=',num2str(maxR)])
stat.edgeMergR = maxR;

% plot the results.
figure; orient tall;
% show an original image
[NonfiltOrg,map] = feval('imread',fn.corg,'bmp');
NonfiltOrg = remBndaryOne(NonfiltOrg,marg);
subplot(321); imshow(NonfiltOrg); title('original Image');
subplot(322);imshow(mat2gray(textMergReg)); title('text region');
subplot(323); imshow(coloredOrg);title('small region removal');
subplot(324); imshow(mat2gray(edgeNTextReg));
title('overlaying edges on nontexture region(nonblack)');
subplot(325);imshow(coloredEdge);title('after splitting regions');
str = strcat('# of max Regs =',num2str(length(unique(NTELbl)))); ylabel(str);
subplot(326);
coloredmergLbl = ind2rgb(mergLbl,[0 0 0 ; jet(max(mergLbl(:)))]);
imshow(coloredmergLbl); ttext = strcat('merged region using edges with ',comment);
title(ttext);
maxElblM = num2str(max(mergLbl(:)));
str = strcat('# of max Regs =',maxElblM); ylabel(str);

% save the result.
if(strcmp(func,'efunc1'))
   sf = 'F1';
else
   sf = 'F2';
end
fnameExt = strcat(fn.seq,'-',sf,'-',num2str(THedge.alpha),'-',num2str(THstat.text));
feval('chdir',savePath);
%feval('saveas',gcf,fnameExt,'fig');
feval('print','-dpsc',fnameExt);
feval('save',fnameExt,'mergLbl','stat');
feval('chdir',curPath);
close all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [AvgRegLength, maxLeng, minLeng, NR] = regLengthCal(LabelImg,conComp)
% function [AvgRegLenth, NR] = regLengthCal(LabelImg,conComp)
```

```
% get the labeled region statics (avg. region length and # of regions) from a label image.
% used for HW region access time.
% Copyright (c) 2000 by JinSang Kim, CSU EE.

stats = imfeature(LabelImg,'BoundingBox',conComp);
[NR,dummy] = size(stats);
regLength = [];
for i = 1: NR
   BoundBox = getfield(stats,{i,1},'BoundingBox');
   regLength = [regLength; BoundBox(:,4)]; % 4th column is the region length.
end
AvgRegLength = mean(regLength);
maxLeng = max(regLength);
minLeng = min(regLength);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ textMask ] = colTextMaskGenOnlyConf(THeig,THsmall,filtEIG, dim, marg)
%
%  Mask generation for texture and non texture regions by only using conf.value.
% 1's are the
%  Copyright (c) 1999 by Jinsang Kim, CSU/ECE.
%
%-----------------------------------------------------------------------
filtEIG = reshape(filtEIG,dim(1,1),dim(1,2));
[filtEIG] = remBndaryOne(filtEIG,marg);
Mask  = roicolor(filtEIG,THeig,max(filtEIG(:)));

% delete small region if requested.
if(~isempty(THsmall))
   %textMask = rmSmallBwNoBndy(Mask,THsmall);
   textMask = rmSmallBw(Mask,THsmall);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ELbl]= regSplitEdgeEZero(edge,conComp,Reg)
%
% [ELbl]= regSplitEdge(edge,conComp,Reg)
%         split the regions(Reg) broken by edges(edge). value 0's are edges.
%         only works on the nonzero label of Reg(ignore background).
%         conComp is connected component.
%
% Copyright (c) 1999 by JinSang Kim ECE/CSU.

% overwrite the edge over nontexture regions and split the region broken by edge.
[r,c] =size(Reg);
maxR = max(Reg(:));
RegR = reshape(Reg,r*c,1);
ELblR = zeros(r*c,1); % contours of Non texture regions
edgeR = reshape(edge,r*c,1);
maxEdgeR = 0;
figure; imshow(edge);
for rl = 1:maxR  % exclude background, texture mask.

   % identify each subregions without edges which can not contribute the regions splitting.
   tmpBwImgR = zeros(r*c,1);
   tmpEdgeR = zeros(r*c,1);
   [i,dummy] = find(RegR == rl); % i is the pixel location of original region.
   tmpEdgeR(i,1) = edgeR(i,1); tmpEdge = reshape(tmpEdgeR,r,c);
   [L,N] = bwlabel(tmpEdge,conComp);

   % L is the subregions map containing edge pixels and,
   % so label them after finding edge locations.
   [ei,dummy] = find(reshape(L,r*c,1)==0); % background and edges.
   subEdges = intersect(i, ei); % only edges in this subregion.
```

```
    LR = reshape(L,r*c,1);

   %  edge pixel stuffing with neighbor label.
    if(~isempty(subEdges))
      NE = length(subEdges);
      for e = 1: NE
       % find neighbors of each edge pixel.
         bw = zeros(r*c,1); bw(subEdges(e),1) = 1; bw=reshape(bw,r,c);
         neigh = bwneighp(bw,conComp); % 1 means neighbors.
         neighp = find(reshape(neigh,r*c,1)>0);
            tmpLbl = LR(neighp,1); % choose non zero minimum value.
         tmpLblGTZero = find(tmpLbl>0);
         if(~isempty(tmpLblGTZero))
           LR(subEdges(e),1) = min(tmpLbl(tmpLblGTZero));
         end
      end
    end

   % relabel regions
   [iLR,dummy] = find(LR>0); % background should be always 0.
   LR(iLR,1) = LR(iLR,1) + maxEdgeR;
   maxEdgeR = (N+maxEdgeR);
   ELblR(iLR,1) = LR(iLR,1);
end
ELbl = reshape(ELblR,r,c);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [mergELbl, stat] = colMergWithEdge(edge,ELbl,highEdge,colFv,...
                                          THedge,func,conComp,stat,THsmall)
%   [mergELbl] = colMergWithEdge(edge,ELbl,highEdge,colFv,THedge,func,conComp)
% merge region splitted by edges.
% INPUTS
%     edge            edge(0 values) linked map.
%     ELbl            regions splitted by edges(zero values means other mask).
%     highEdge        edge lined map with high threshold.
%     colFv               feature vector.(should be only feature on this mask).
%     THedge      threshold values for merging.
%     func            energy function, 'efunc1' or 'efunc2'
%
% OUTPUTS
%     mergELbl        merged output.
%
% ALGORITHM
% while(# of merged regions >0)
%   for i=1 smallest region to largest.
%     - exclude neighbors with "weak geometric connection".
%   - compute the energy and choose the min of j.
%     - Eij = (feature distance)ij + alpha*(boundary strength)ij.
%             Eij and alpha ={ positive, if boundary is strong.
%                            { negative,      "          weak.
%     - merge the region i and j if Eij <0.
%   end
% end
% Copyright (c) 1999 by JinSang Kim, CSU EE.

[r,c] = size(ELbl);
NMERG = 1;    % dummy.
NWGC = 0; TNEIG = 0; NWBND = 0;
MAXNR = 0;  NLOOP = 0;

while(NMERG)
   % region statistics ( region box length).
   'nontextured region (edge fusion) length statics'
   % get the labeled region statics for nontextured regions. (region length).
```

```
% used for HW region access time.
[AvgRegLenth, maxLeng, minLeng, NR] = regLengthCal(ELbl,conComp)

% sort regions again.
skipZero = 1;  % skip Zero label for sorting(background).
index = sortSml2Lrg(ELbl,skipZero); % index is the label order.

NREG = length(index);
NMERG = 0;
NACTMERGREG = 0; % the number of actual merged region counters per loop.
TACTMERGREGRL = 0; % total region length of actual merged regions.

for i= 1:NREG
   % just select neghbors with "strong geometric connections".
   [neigLblSGC,tmpNWGC] = findStrConNeig(ELbl,index(i),THedge.sgc,conComp);
   NWGC = NWGC + tmpNWGC;
   TNEIG = TNEIG +length(neigLblSGC);
   MAXNR = max(MAXNR,length(neigLblSGC));

   if(~isempty(neigLblSGC))
     % compute boundary strength.
     if(~isempty(highEdge)) % if specifed, use highedge for edge strenth.
        edge = highEdge;
     end

     [boundStr,tmpNWBND] = bndyStrg(ELbl,edge,index(i),neigLblSGC,conComp,THedge.beta);
     NWBND = NWBND + tmpNWBND;

     % compute feature mean distance.
    featDist_Y = featureMeanDist(ELbl,colFv(:,1),index(i),neigLblSGC);%Y component.
     featDist_U = featureMeanDist(ELbl,colFv(:,2),index(i),neigLblSGC);%U component.
     featDist_V = featureMeanDist(ELbl,colFv(:,3),index(i),neigLblSGC);%V component.

     % compute the energy and choose the min. of j. column vector.
    Eij_Y = feval(func,featDist_Y,boundStr,THedge.alpha,THedge.beta);
     Eij_U = feval(func,featDist_U,boundStr,THedge.alpha,THedge.beta);
        Eij_V = feval(func,featDist_V,boundStr,THedge.alpha,THedge.beta);

     % find indices for all zero energies.
     negaYind = find(Eij_Y<=0);
     negaUind = find(Eij_U<=0);
        negaVind = find(Eij_V<=0);
     negaYUVind = intersect(intersect(negaYind,negaUind),negaVind);

     % calculate current region size.
     curBw = roicolor(ELbl,index(i),index(i));
     curBwArea = bwarea(curBw);

     % merg if Eij is negative, choose minimun energy.
     ELblR = reshape(ELbl,r*c,1);
     if(~isempty(negaYUVind))
       % calculate # of merged region & size.
       NACTMERGREG = NACTMERGREG + 1;
       RL = regLengthCalOneReg(ELbl,conComp,index(i));
       TACTMERGREGRL= TACTMERGREGRL + RL;

       % find smallest energy as a merging region.
       Eij =[Eij_Y Eij_U Eij_V];
       Eij_nega = Eij(negaYUVind,:);
       [dummy,minIndOfNega] = min(sum(Eij_nega'));
       mergedLbl = neigLblSGC(negaYUVind(minIndOfNega));
       % merge two regions.
       ELblR = merge2Reg(ELblR,index(i),mergedLbl);
       NMERG = NMERG+1;
```

```
          elseif(curBwArea<THsmall)
             Eij =[Eij_Y Eij_U Eij_V];
             [dummy,minInd] = min(sum(Eij'));
             mergedLbl = neigLblSGC(minInd);
            % merge two regions.
             ELblR = merge2Reg(ELblR,index(i),mergedLbl);
             NMERG = NMERG+1;
         end
         ELbl = reshape(ELblR,r,c);
      else
         ELblR = reshape(ELbl,r*c,1);
         ELbl = reshape(ELblR,r,c);
      end

    end

    'NLOOP and NREG'
    NLOOP = NLOOP + 1
    NREG
   'The number of regions of actually merged current regions and Avg. Region Length (RL)'
    NACTMERGREG
    ACTMERGRRLAVG = TACTMERGREGRL / NACTMERGREG

  end
  ELbl = getLabelFromRegionEZero(ELbl,conComp);
  mergELbl = ELbl;
  stat.wgcPer = [];
  stat.wbndPer = [];
  if(TNEIG ~=0)
   stat.wgcPer = NWGC/TNEIG;
   stat.wbndPer = NWBND/TNEIG;
  end

  'Max. Neig. Region.'
  MAXNR
   index = sortSml2Lrg(ELbl,skipZero); % index is the label order.
   NREG = length(index);
   maxLabel = index(length(index))
  % find binary image for maxLabel.
   ELblR = reshape(ELbl,r*c,1);
   [i,dum] = find(ELblR == maxLabel);
   MAXRS = length(i);
   tmp = zeros(r*c,1);
   tmp(i,1) = 1;
   tmp = reshape(tmp,r,c);
   figure;imshow(tmp);
   bwRP = bwperim(tmp,4);
   bwRPR = reshape(bwRP,r*c,1);
   [k,dum] = find(bwRPR == 1);
   MAXRP = length(k);
   'max region size'
   MAXRS
   'max perimeter'
   MAXRP

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [neigSGC, NWGC]= findStrConNeig(reg,curLbl,THsgc,conComp)
% [neigLbl]= findStrConNeig(reg,curLbl,neigLbl,THwgc)
%      exclude neighbors with "weak geometric connections" by THwgc.
%     also, exclude 0 label neighbor(background, other MASK).
%     Cij/min(Pi,Pj)
% Copyright (c) 1999 by JinSang Kim, CSU EE.
```

```
[r,c] = size(reg);
regR = reshape(reg,r*c,1);

% current region
curRegBw = roicolor(reg,curLbl,curLbl); % current regions.
bwnp = bwneighp(curRegBw,conComp); % BW neighbor regions.
bwnpr = reshape(bwnp,r*c,1);
[i1,j1] = find(bwnpr);

% find neighbor regions except 0 label.
Llbls = regR(i1,:);
neigLbl = unique(Llbls); % # of negibhor.
[k,l] = find(neigLbl);
neigLbl = neigLbl(k);

NWGC = 0;
neigSGC = [];
if(~isempty(neigLbl))
   % count pixels on common boundaries.(Cin).
   % determine the pi and pj(j is neighbor of i). use outside boundaries.
   Pi = length(i1);
   NNEIG = length(neigLbl);
   for i = 1:NNEIG
   [val,index] = find(Llbls == neigLbl(i));
   cin(i) = length(index);
   neigRegBw = roicolor(reg,neigLbl(i),neigLbl(i));
   neigNpBw = bwneighp(neigRegBw,conComp); [ci,di] = find(neigNpBw == 1); Pj(i) = length(ci);
   minPeri(i) = min([Pi Pj(i)]); % actually outside region boundaries.
   cinRatio(i) = cin(i)/minPeri(i);
   end
   % exclude neighbors whose ratio is less than THwgc.
  [dummy,index] = find(cinRatio >= THsgc);
  NWGC = NNEIG - length(index);
  neigSGC = neigLbl(index);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [boundStr,NWBND]= bndyStrg(reg,edge,curLbl,neigLblSGC,conComp,beta)
%  boundStr = bndyStrg(reg,edge,curLbl,neigLblSGC)
%       calculate boundary strength between curLbl and neigLblSGC.
%     Eij/Cij, to calculate the Eij(# of edges on the bondaries).
%     check the edges on the common perimeter and outside perimeters.
%       choose larger one as a result.
%  Copyright (c) 1999 by JinSang Kim, CSU EE.

[r,c] = size(reg);
regR = reshape(reg,r*c,1);
edgeR = reshape(edge,r*c,1);

% current region
curRegBw = roicolor(reg,curLbl,curLbl); % current regions.
bwnp = bwneighp(curRegBw,conComp); % BW neighbor regions.
bwnpr = reshape(bwnp,r*c,1);
[i1,j1] = find(bwnpr);

% find neighbor regions.
Llbls = regR(i1,:); % i1 is the neighbor position.

% count pixels on common boundaries.(Cin).
% determine the pi and pj(j is neighbor of i). use outside boundaries.
NNEIG = length(neigLblSGC);
cij = [];
for i = 1:NNEIG
  [val,index] = find(Llbls == neigLblSGC(i));
```

```
  cij(i) = length(index);  % # of pixels for ith neighbor.
  % find the ith neighbor's pixel position(column vector).
  [nposition,dummy] = find(regR(i1) == neigLblSGC(i));
  npi = i1(nposition); % position of a neighbor, columnized.

  % find Eij(# of edges )on ith neighbor.
  subEdge = edgeR(npi); % edges on Cij.
  if(~isempty(subEdge))
   [ei,dummy] = find(subEdge == 0);
   eij(i) = length(ei);
  else
     eij(i) = 0;
   end
end

boundStrOut = (eij ./ cij);
[k,l] = find(boundStr<beta);
NWBND = length(k);
```

Bibliography

[1] Jinsang Kim and Tom Chen, "Integation of Multiple Features Using Back Propagation Neural Networks for Segmentation of Image Sequences," in *Int. Conference on Imaging Science, System and Technology, Las Vegas, Nevada, USA*, June 1999, pp. 65–71.

[2] Jinsang Kim and Tom Chen, "Multiple Feature Clustering for Image Sequence Segmentation," *Pattern Recognition Letters (accepted)*, 2000.

[3] MPEG Group, "Mpeg-4 video verification model," 1999.

[4] A. Tekalp, *Digital Video Processing*, Prentice Hall, 1995.

[5] Luis Torres and Murat Kunt, Eds., *Video Coding - The Second Generation Approach*, Kluwer Academic Publishers, 3rd edition, 1996.

[6] Chunag Gu and Ming-Chieh Lee, "Semiautomatic Segmentation and Tracking of Semantic Video Objects," *IEEE Trans. CSVT*, vol. 8, pp. 572–584, Sep. 1998.

[7] Roberto Castagno and et al., "Video Segmentation Based on Multiple Features for Interactive Multimedia Applications," *IEEE Trans. CSVT*, vol. 8, pp. 562–571, Sep. 1998.

[8] A. Aydin Alatan and et al., "Image Sequence Analysis for Emerging Interactive Multimedia Services - The European COST 211 Framework," *IEEE Trans. CSVT*, vol. 8, pp. 802–813, Nov. 1998.

[9] MPEG Requirement Group, "Mpeg-4 overview," 1999.

[10] Aggelos K. Katsaggelos, Lisimachos P. Kondi, Pabian W. Meier, Jorn Ostermann, and Guido M. Schuster, "MPEG-4 and Rate-Distortion-Based Shape-Coding Techniques," *Proceedings of the IEEE*, vol. 86, pp. 1126–1154, 1998.

[11] Paul R. Cohen and Edward A. Feigenbaum, *The handbook of artificial intelligence: Vol. III*, William Kaufmann Inc., Los Altos, California, USA, 1982.

[12] J. Hadamard, *Lectures on the Cauchy Problem in Linear Partial Differential Equations*, Yale University Press, 1923.

[13] G. Adiv, "Determining three-dimensional motion and structure from optical flow generated by several moving objects," *IEEE Trans. PAMI*, vol. 7, pp. 384–401, July 1985.

[14] J. M. Odobez and P. Bouthemy, "Robust multiresolution estimation of parametric motion models," *Journal of visual communication and image representation*, vol. 6, pp. 348–365, Dec. 1995.

[15] John Y. A. Wang and Edward H. Adelson, "Representing Moving Images with Layers," *IEEE Trans. IP*, vol. 3, pp. 625–638, 1994.

[16] Heyun Zheng and Steven D. Blostein, "Motion-Based Object Segmentation and Estimation Using the MDL Principle," *IEEE Trans. IP*, vol. 4, pp. 1223–1235, Sep. 1995.

[17] Michael M. Chang, A. Murat Tekalp, and M. Ibrahim Sezan, "Simultaneous Motion Estimation and Segmentation," *IEEE Trans. IP*, vol. 6, pp. 1326–1333, Sep. 1997.

[18] Di Zhong and Shih-Fu Chang, "An Integrated Approach for Content-Based Video Object Segmentation and Retrieval," *IEEE Trans. on CSVT*, vol. 9, pp. 1259–1268, Dec. 1999.

[19] Munchurl Kim and et al., "A VOP Generation Tool: Automatic Segmentation of Moving Objects in Image Sequences Based on Spatio-Temporal Information," *IEEE Trans. on CSVT*, vol. 9, pp. 1216–1226, Dec. 1999.

[20] P. Erhan Eren, Yucel Altunbasak, and A. Murat Tekalp, "Region-based affine motion segmentation using color information," in *IEEE Int. Conf. ASSP*, 1997, pp. 3005–3008.

[21] Yining Deng and B. S. Manjunath, "NeTra-V: Toward an Object-Based Video Representation," *IEEE Trans, CSVT*, vol. 8, pp. 616–627, Sep. 1998.

[22] M. Kunt, A.Ikonomopoulos, and M. Kocher, "Second generation image coding techniques," *Proceedings of the IEEE*, vol. 73, pp. 549–575, April 1985.

[23] Motion Picture Expert Group, "Information technology - coding of moving pictures and associated audio for digital storage media at up to about 1.5 mbit/s-part 2: Video," 1993.

[24] Motion Picture Expert Group, "Information technology - generic coding of moving pictures and associated audio information: Video," 1996.

[25] ITU, "Itu-t recommendation h.261 - video codec for audiovisual services at p×64 kbit/s," March 1993.

[26] ITU, "Itu-t recommendation h.263 - video coding for low bit rate communication," March 1993.

[27] B.Girod, *Motion compensation: visual aspects, accuracy and fundamental limits. in M.I.Sezan and R.L. Lagendijk, editors, Motion analysis and image sequence processing, pages 125-152*, Kluwer Academic Publishers, 1993.

[28] MPEG Group, "Report of the formal verification tests on mpeg-4 coding efficiency for low and medium bit rates," July/1999.

[29] The MPEG Home Page, "http://www.cselt.it/mpeg/," .

[30] Tomaso Poggio and et al., "Computational vision and regularization theory," *Nature*, vol. 317, pp. 314–319, 1985.

[31] Gaurav Sharma and H. Joel Trussel, "Digital Color Imaging," *IEEE Trans. on IP*, vol. 6, pp. 901–931, July 1997.

[32] Wyszecki G and Stiles W. S., *Color science*, Wiley, New York, 1982.

[33] Rafael C. Gonzalez and Richard E. Woods, *Digital image processing*, Addison-Wesly, 1992.

[34] A. K. Jain, *Fundamentals of Digital Image Processing*, Prentice Hall, Englewood Cliffs, N.J, 1989.

[35] J. F. Canny, "A computational approach to edge detection," *IEEE Trans. PAMI*, vol. 8, pp. 679–698, 1986.

[36] C. Y. Lee and et al., "An efficient ASIC architecture for real-time edge detection," *IEEE Trans. Circuits and Systems*, vol. 36, pp. 1350–1359, 1989.

[37] L. Torres and et al., "Implementation of A Recursive Real Time Edge Detector Using Retiming Technique," in *VLSI'95*, Tokyo, Japan, 1995, pp. 811–816.

[38] F. Alzahrani and T. Chen, "A Real Time Edge Detector: Algorithm and VLSI Architecture," *Real-Time Imaging*, vol. 3, pp. 363–378, 1997.

[39] Amjad and T. Chen, "A VLSI Architecture for Real-Time Edge Linking," *IEEE Trans. PAMI*, vol. 21, pp. 89–94, 1999.

[40] Nikhil R. Pal and Sankar K. Pal, "A Review on Image Segmentation Techniques," *Pattern Recognition*, vol. 26, pp. 1277–1294, 1993.

[41] K. I. Laws, "Textured image segmentation," Tech. Rep. USCIPI Report940, Dept. of Elec. Eng., Image Processing Institute,Univ. of Southern California, Los Angeles, January 1980.

[42] R. M. Haralick, K. Shanmugam, and I. Dinstein, "Textural features for image classification," *IEEE Trans. Systems Man Cybernet*, vol. 3, pp. 610–621, Jan. 1973.

[43] Dennis Dunn and Willian E. Higgins, "Optimal Gabor Filters for Texture Segmentation," *IEEE Trans. IP*, vol. 4, pp. 947–964, 1995.

[44] Gilbert Strang and Truong Nguyen, *Wavelet and Filter Banks*, Wellesley-Cambridge Press, Wellesley, MA, USA, 1996.

[45] T. Poggio and et al., "Parallel Integration of Vision Modules," *Science*, vol. 242, pp. 436–440, 1988.

[46] B. D. Lucas and T. Kanade, "An iteractive image registration technique with an application to stereo vision," in *Proceedings of DARPA Image Understanding Workshop*, 1981, pp. 121–130.

[47] P. Anandan, "A Computational Framework and an Algorithm for the Measurement of Visual Motion," *International Journal of Computer Vision*, vol. 2, pp. 283–310, 1989.

[48] D. Gabor, "Theory of communication," in *Proc. Inst. Elec. Eng.*, 1946, vol. 93, pp. 429–441.

[49] C. E. Shannon, "Communication in the presence of noise," in *Proc. IRE*, 1949, vol. 37, pp. 10–21.

[50] J. Ross Beveridge and et al., "Segmenting Images Using Localized Histograms and Region Merging," *International Journal of Computer Vision*, vol. 2, pp. 311–347, 1989.

[51] J. S. Weszka, R. N. Nagel, and A. Rosenfeld, "A threshold selection techniques," *IEEE Trans. Computers*, vol. 23, pp. 1322–26, 1974.

[52] J. S. Weszka, R. N. Nagel, and A. Rosenfeld, "A survey of threshold segmentation technique," *Computer Graphics and Image Processing*, vol. 7, pp. 259–265, 1978.

[53] J. S. Weszka, "A survey of threshold segmentation techniqes," *Computer Graphics and Image Processing*, vol. 7, pp. 259–265, 1978.

[54] J. Bryant, "On the clustering of multidimensional pictorial data," *Pattern recognition*, vol. 11, pp. 115–125, 1979.

[55] R. M. Haralick, "Digital step edges from zero crossing of second directional derivative," *IEEE Trans. PAMI*, vol. 6, pp. 58–68, 1984.

[56] P. C. Chen ad T. Pavlidis, "Image segmentation as an estimation problem," *Computer Graphics and Image Processing*, vol. 12, pp. 153–172, 1980.

[57] M. Suk and S. M. Chung, "A new image segmentation technique based on partition test," *Pattern recognition*, vol. 16, pp. 469–480, 1983.

[58] O. J. Morris and M. De J. Lee, "Graph theory for image analysis: an approach based on the shortest spanning tree," *IEE Proceedings*, pp. 146–152, April 1986.

[59] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, 1981.

[60] G. H. Ball and D. J. Hall, "ISODATA, a novel method of data analysis and pattern classification," in *Proceedings of the Int. Communication Conference*, Philadelphia, 1966.

[61] K. S. Fu and J. K. Mui, "A survey on image segmentation," *Pattern Recognition*, vol. 13, pp. 3–16, 1981.

[62] L. G. Shaprio and R. M. Haralick, "Survey, image segmentation techniques," *Computer Vision Graphics Image Processing*, vol. 29, pp. 100–132, 1985.

[63] A. K. Jain, *Algorithms for Clustering Data*, Prentice Hall, 1988.

[64] J. Mao and A. K. Jain, "Texture Classification and Segmentation using Multiresolution Simultaneous Autoregressive Models," *Pattern Recognition*, vol. 25, pp. 173–188, 1992.

[65] Steven M. Kay, *Fundamentals of statistical signal processing:estimation theory*, Prentice Hall, New Jersey, 1993.

[66] Nuno Vasconcelos and Andrew Lippman, "Empirical Bayesian EM-based Motion Segmentation," in *IEEE conf. CVPR'97*, San Juan, Puerto Rico, 1997.

[67] G. R. Cross and A. K. Jain, "Markov random field texture models," *IEEE Trans. PAMI*, vol. 5, pp. 25–39, 1983.

[68] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distribution, and Bayesian restoration of images," *IEEE Trans. PAMI*, vol. 6, pp. 721–741, 1984.

[69] H. Derin and H. Elliott, "Modeling and segmentation of noisy and textured images using Gibbs random field," *IEEE Trans. PAMI*, vol. 9, pp. 39–55, 1987.

[70] E. Dubois and J. Konrad, *Estimation of 2-D motion fields from image sequences with application to motion-compensation processing*, in Motion Analysis and Image Sequence Processing, M. I. Sezan and R. L. Lagendijk eds, Kluwer, Norwell, MA, 1993.

[71] J. Besag, "Spatial interaction and the statistical analysis of lattice systems," *J. Royal Stat. Soc.*, vol. 36, pp. 192–236, 1974.

[72] F. Meyer and S. Beucher, "Morphologcal Segmentation," *Journal of Visual Communication and Image Representation*, vol. 1, pp. 21–46, 1990.

[73] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing Company, 1994.

[74] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 4, pp. 59–69, July 1982.

[75] J. Y. Wang and E. H. Adelson, "Representing Moving Images with Layers," *IEEE Trans, IP*, vol. 3, pp. 625–638, Sep. 1994.

[76] J. Hutchinson, C. Koch, J. Luo, and C. Mead, "Computing motion using analog and binary resistive networks," *Computer*, vol. 21, pp. 52–63, Mar. 1988.

[77] Fabrice Moscheni and et al., "Spatiotemporal Segmentation Based on Region Merging," *IEEE Trans. PAMI*, vol. 20, pp. 897–915, 1998.

[78] Demin Wang, "Unsupervised Video Segmentation Based on Watersheds and Temporal Tracking," *IEEE Trans. CVST*, vol. 8, pp. 539–546, 1998.

[79] F. Dufaux, F. Moscheni, and A. Lippman, "Spatio-temporal segmentation based on motion and static segmentation," in *IEEE Proc. ICIP'94*, Austin, TX, USA, 1994, vol. 2, pp. 428–432.

[80] F. Moscheni, F. Dufaux, and M. Kunt, "A new two-stage global/local motion estimation based on a background/forground segmentation," in *IEEE Proc. ICASSP'95*, Detroit, MI, USA, 1995.

[81] L. Kaufman and P.J. Rousseeuw, *Finding Groups in Data: an Introduction to Cluster Analysis*, Wiley, New York, 1990.

[82] D. Zhong and S. F. Chang, "Video Object Model and Segmentation for Content-Based Video Indexing," in *IEEE Proc. ISCAS'97*, Hong Kong, 1997.

[83] Thomas Meier and King N. Ngan, "Automatic Segmentation of Moving Objects for Video Object Plane Generation," *IEEE Trans. CSVT*, vol. 8, pp. 525–538, 1998.

[84] Kevin W. Bowyer and P. Jonathon Phillips, *Empirical Evaluation Techniques in Computer Vision*, IEEE Computer Society, Los Alamitos, CA, USA, 1998.

[85] Y. J. Zhang, "A Survey on Evaluation Methods for Image Segmentation," *Pattern Recognition*, vol. 29, pp. 1335–1346, 1996.

[86] Adam Hoover and et al., "An Experimental Comparison of Range Image Segmentation Algorithm," *IEEE Trans. PAMI*, vol. 18, pp. 673–689, July 1997.

[87] C. E. Liedtke, T. Gahm, F. Kappei, and B. Aeikens, "Segmentation of microscopic cell scenes," *AQCH*, vol. 9, pp. 197–211, 1987.

[88] C. Garbay, "Image structure representation and processing: a discussion of some segmentation methods in cytology," *IEEE Trans. PAMI*, vol. 8, pp. 140–146, 1986.

[89] M. D. Levine and A. Nazif, "Dynamic measurement of computer generated image segmentations," *IEEE Trans. PAMI*, vol. 78, pp. 155–164, 1985.

[90] Jianqing Liu and Yee-Hong Yang, "Multiresolution Color Image Segmentation," *IEEE Trans. PAMI*, vol. 16, pp. 689–700, 1994.

[91] W. A. Yasnoff, J. K. Mui, and J. W. Bacus, "Error measures for scene segmentation," *Pattern Recognition*, vol. 9, pp. 217–231, 1977.

[92] Jinsang Kim and Tom Chen, "Combining Static and Dynamic Features Using Neural Networks and an Edge Fusion for Image Sequence Segmentation," *IEEE Trans. on PAMI (submitted, log number=112453)*, 2000.

[93] Fabrice Moscheni, Sushil Bhattacharjee, and Murat Kunt, "Spatiotemporal Segmentation Based on Region Merging," *IEEE Trans. on PAMI*, vol. 20, pp. 897–915, Sep. 1998.

[94] J. Ohm and P. Ma, "Feature-Based Cluster Segmentation of Image Sequences," in *IEEE Int. Conf. IP*, 1997, pp. 178–181.

[95] Teuvo Kohonen, "The Self-Organizing Map," in *Proceedings of the IEEE*, Sep. 1990.

[96] B. S. Everitt, *Cluster Analysis*, John Wiley & Sons Inc., New York, 3rd edition, 1993.

[97] J. Shi and C. Tomasi, "Good feature to track," in *IEEE Int. Conf. CVPR*, 1994, pp. 593–600.

[98] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of Optical Flow Techniques," *Int. J. of Computer Vision*, vol. 12, pp. 43–77, 1994.

[99] S. Y. Kung and J. N Hwang, "Neural Networks for Intelligent Multimedia Processing," in *Proceedings of the IEEE*, June 1998, pp. 1244–1272.

[100] Jinsang Kim and Tom Chen, "Multiple Feature/Neural Network based Segmentation for Image Sequences," in *Statistical Methods for Image Processing, A satellite conference of the 52nd ISI Session, Upssala, Sweden*, Aug. 1999.

[101] Jinsang Kim and Tom Chen, "Segmentation of Image Sequences Using SOFM Networks," in *15th International Conference on Pattern Recognition, Barcelona, Spain*, Sep. 2000.

[102] Jinsang Kim and Tom Chen, "Neural Network Based Image Sequence Segmentation Using Multiple Features and Edge Fusion," in *Advanced Concepts for Intelligent Vision Systems Symposium, Baden-Baden, Germany*, July 2000.

[103] Jinsang Kim and Tom Chen, "An Integrated Approach to Image Sequence Segmentation," in *IEEE Nordic Signal Processing Symposium, Vildmarkshotellet, Sweden*, June 2000.

[104] Jinsang Kim and Tom Chen, "Low-Complexity Fusion of Intensity, Motion, Texture and Edge for Image Sequence Segmentation: A Neural Network Approach," in *IEEE International Workshop on Neural Networks for Signal Processing, Sydney, Australia*, Dec. 2000.

[105] Marie Pierre Dubuisson and Anil K. Jain, "Contour Extraction of Moving Objects in Complex Outdoor Scenes," *International Journal of Computer Vision*, vol. 14, pp. 83–105, 1995.

[106] Dorin Comaniciu and Peter Meer, "Robust Analysis of Feature Spaces: Color Image Segmentation," in *IEEE conf. CVPR'97*, San Juan, Puerto Rico, 1997.

[107] Leonardo Chiariglione, "MPEG and Multimedia Communications," *IEEE Trans. CVST*, vol. 7, pp. 5–18, 1997.

[108] T. Sikora, "The MPEG-4 Video Standard Verification Model," *IEEE Trans. CSVT*, vol. 7, pp. 19–31, Feb. 1997.

[109] MPEG Group, "Mpeg-7 overview (version 2.0)," 1999.

[110] Behrooz Parhami, *Computer Arithmetic - Algorithms and Hardware Designs*, Oxford University Press, New York, USA, 1999.

[111] Kesiab K. Parhi, *VLSI Digital Signal Processing Systems - Design and Implementation*, John Wiley & Sons, New York, USA, 1999.

[112] TMS320C6000, "http://www.ti.com/sc/docs/products/dsp/index.htm," .

[113] ADSP-2100 Family, "http://www.analog.com/industry/dsp/overview/ general.html#adsp," .

[114] Peter Pirsch and Hnas-Joachim Stolberg, "VLSI Implementations of Image and Video Multimedia Processing Systems," *IEEE Trans. on CVST*, vol. 8, pp. 878–891, Nov. 1998.

[115] Omid Gatemi and Sethuraman Panchanathan, "Fractal Engine: An Affine Video Processor Core for Multimedia Applications," *IEEE Trans. on CVST*, vol. 8, pp. 892–908, Nov. 1998.

[116] N. Ranganathan and R. Mehrotra, "A VLSI Architecture for Dynamic Scene Analysis," *Computer Vision, Graphics, and Image Processing: Image Understanding*, Mar. 1991.

[117] Piotr Wailewski, "Hardware implementation of image segmentation algorithm for real-time image compression," in *Proceedings of SPIE - the international society for optical engineering*, 1998, pp. 106–114.

[118] K. V. Asari, T. Srikanthan, S. Kumar, and D. Radhakrishnan, "A pipelined architecture for image segmentation by adaptive progressive thresholding," *Microprocessors and Microsystems*, vol. 23, pp. 493–498, 1999.

[119] W. F. Blanz, C. B. Shung, C. E. Cox, W. Greiner, B. E. Dom, and D. Petkovic, "Design and implementation of a low-level image segmentation architecture - LISA," *Machine Vision and Applications*, vol. 6, pp. 181–190, 1993.

[120] Jinsang Kim and Tom Chen, "A VLSI Architecture for Image Sequence Segmentation Using Edge Fusion," in *International Workshop on Computer Architectures for Machine Perception, Padova, Italy*, Sep. 2000.

[121] Jinsang Kim and Tom Chen, "A VLSI Architecture for Image Sequence Segmentation," *IEEE Trans. on PAMI (submitted, log number=112454)*, 2000.

[122] H. T. Kung, "Why Systolic Architecture?," *IEEE Computer*, pp. 37–46, Jan. 1982.

[123] David Patterson and John Hennessy, *Computer Architecture - A Quantitative Approach*, Morgan Kaufmann Publishers, San Francisco, USA, 1996.

[124] S. B. Pan, S. S. Chae, and R. H. Park, "VLSI Architectures for Block Matching Algorithms Using Systolic Arrays," *IEEE Trans. on CSVR*, vol. 6, pp. 67–73, Feb. 1996.

[125] Mustafa Karaman, Levent Onural, and Abdullah Atalar, "Design and Implementation of a General-Purpose Median Filter Unit in CMOS VLSI," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 505–513, April 1990.

[126] Xue Dong Yang, "Design of Fast Connected Components Hardware," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1996, pp. 937–944.

[127] Clark D. Thompson, "The VLSI Complexity of Sorting," *IEEE Trans. on Computers*, vol. 32, pp. 1171–1184, 1983.