Algorithms for Video Object Detection and Segmentation with Application to Content-Based Multimedia Systems

Huitao Luo

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

in the Graduate School of Arts and Sciences

Columbia University

2000

© 2000

Huitao Luo

All Rights Reserved

ABSTRACT

Algorithms for Video Object Detection and Segmentation with Application to Content-Based Multimedia Systems

Huitao Luo

We have designed several analysis algorithms for video object (VO) detection and segmentation, which is an important part of the new MPEG4 and MPEG7 standards. The algorithms are developed in two directions, *i.e.*, automatic algorithms and semi-automatic algorithms.

In automatic analysis research, our work is mainly focused on model-based algorithm design. Instead of working on general-purpose algorithms, we design detection and segmentation algorithms for typical multimedia applications. By constraining the application domain, we could represent the video objects to be processed with some *models*. The models are created with specific statistical structures, but are still generally applicable in certain application domains. Specifically, we have designed video object detection and/or segmentation algorithms for three applications: (1) Real-time VO segmentation for videophones. (2) Anchorperson detection and segmentation for broadcast news indexing and retrieval. (3) Face detection in the compressed DCT domain.

For videophone applications, we design a blob-based region model and a shape model to represent typical *head-and-shoulder* foreground. Both models use a Gaussian assumption for their feature vectors. At the system level, a hierarchical structure is designed to support an online processing of model-creating, model-fitting, and model-updating. In our experiments, a QCIF size video sequence is segmented in real time using software only into three video objects: a background, a head and a

shoulder on average Pentium PC platforms. Based on the real time performance of the algorithm, we discuss two direct applications of it, *i.e.*, real time VO generation for MPEG-4 codecs and content-based bit-rate control for traditional H.263 codecs.

For anchorperson detection, we propose to model anchorperson patterns with their color and shape features. The detection problem is decomposed into a color model based face region detection and a shape model based head-and-shoulder pattern detection problem. The statistical shape model design is similar to what we used for videoconference applications, except an offline model fitting.

Our work on face detection proposes to merge two types of face detection algorithms in the literature, skin-color based and texture based face detection. Though each of them has been explored extensively, our work for the first time shows that they can be combined with a hybrid statistical model (color-texture model) to generate better detection performance. In addition, the hybrid model is designed in the compressed DCT domain. A number of fundamental problems, *e.g.*, block quantization problem, preprocessing, and feature vector selection and classification in the DCT domain, are discussed.

In semi-automatic analysis research, an interactive authoring system is designed for video object segmentation. This system features a new contour interpolation algorithm, which enables the user to define the contour of a VO on multiple anchor frames while the computer interpolates the missing contours of this object on every frame automatically. Typical active contour model is adapted and the contour interpolation problem is decomposed into two directional contour tracking problems and a merging problem. In addition, new user interaction models are created for the user to interact with the computer. Experiments indicate that this system offers a good balance between algorithm complexity and user interaction efficiency.

Contents

List of Figures

List of Tables

Acknowledgements

I would first express my sincere gratitude to Prof. Alexandros Eleftheriadis for sponsoring and supervising my research work. During my four years' staying at Columbia, Alex has been a mentor as well as a friend to me. I especially thank him for his consistent support and encouragement, both academically and morally, without which I could not have finished this dissertation.

As a member of our Advent Group, I have benefited a lot from interacting with its members. I thank in particular Prof. Shih-Fu Chang and Prof. Dimitris Anastassiou for promoting an encouraging academic environment within the group, Sara Brock for maintaining the smooth group operation and Bill Paul for helping me with a number of computer problems.

I am also grateful to Dr. Jack Kouloheris of IBM T.J.Watson Research Center and Dr. Qian Huang of AT&T Research Labs for giving me the opportunity to spend two fruitful summers with them. A number of research projects I worked on there have influenced the development of this dissertation.

Among my friends at Columbia, many of them have made my stay more enjoyable. Ching-Yung Lin has been my officemate ever since we started the graduate study together. I have always had his company in the office, being late in the night or in the weekends. Louis Wang and Fan Zhang have been my close friends who helped me to settle down the first day I came to Columbia, and who invited me to their weekend dinners from time to time. I had also many helpful research discussions with Dr. Jae-Beom Lee and Di Zhong.

At the last but not the least, I would like to thank my parents and sister who have loved and encouraged me through all these years. I love you all.

Chapter 1

Introduction

1.1. History and Motivation

The emergence and development of modern multimedia technology mainly happened within the past 5-10 years. One or two decades ago, people were still enjoying analogue telephones, black and white televisions and simple cassette recorders. The major use of computers was for scientific computing. There was no concept of multimedia systems. But today, powerful personal computers with high speed Internet connections, Hi-Fi CD based audio systems, VCD and DVD players are all quite common in average households. The concept of multimedia is not only a hot topic in the research community, but also influencing the way of people's life.

The fundamental driving forces behind these changes are the fast advancing in the computing, communication and storage techniques and industries. Especially in the past five years, the processing power of personal computers has been experiencing exponential climbing. It is no long difficult to decode one or even several MPEG videos in real time on a good Intel architecture personal computer without any hardware acceleration. The solving of even challenging tasks such as real time encoding of MPEG video, real time rendering of 3D texture, *etc.*, are already on the near horizon. In communication industry, wide-bandwidth optical fibers constitute

powerful backbone network, which supports Internet service to the every corner of the world. People send and receive emails, make phone calls with voice over IP, download video streams from news web sites such CNN, MSNBC, *etc.* Even videophones are becoming popular with the price drop of CCD cameras and more popular use of 56K modem and DSL techniques. In the storage industry, high capacity storage devices are becoming more reliable and affordable. A single piece of DVD disk can hold two hours' high quality digital video and it is also possible to add high capacity hard disks to traditional TV set top boxes to record TV programs and offer better services such as browsing, fast forwarding, replaying, *etc.*

With all these developments, the state-of-the-art of multimedia research is also experiencing big changes. In the 1980s, voice was the only media that the capacity of our communication channels could accommodate. Therefore voice compression and coding was the major research issue at that time. Typical technical achievements at that time include DPCM, VQ and subband coding. In the early 1990s, with the maturity of personal computers and better communication capacities, digital image and video became the media forms that people were eager to handle. Intensive research efforts were directed to the efficient compression and coding of digital image and especially digital video. Fruitful results were obtained both theoretically and practically. In the theoretical aspect, traditional source coding framework was built up solidly. Statistical coding methods such as Huffman coding, Arithmetic coding were thoroughly studied. The capacities of different kinds of transform coding methods, especially discrete cosine transform (DCT), wavelet and subband transforms were also compared and investigated in detail. In addition, because of the need of video coding, a complete theoretical system was built up for motion analysis, which includes motion estimation, motion compensation and motion segmentation. In the application aspect, a series of international standards were built up that represent

the good balance between algorithm efficiency and system realization complexity. These include: JPEG for coding continuous tone images, JBIG for coding bi-level images, MPEG-1 for coding digital video for the purpose of storage and playback, MPEG-2 for coding of digital video (mainly) for the purpose of broadcasting, and H.261 and H.263 for real time videophones and videoconferencing service. The availability of these standards helped the convergence in multimedia industry and promoted the commercialization of research results. The efforts of standardization were so successful that the MPEG working group was awarded the EMMY AWARD [15] in 1996.

Technically, the major topic of early multimedia research, including the related standards was to compress the large amount of media data (especially video). It is reasonable to mention the standards such as JPEG, MPEG-1 and MPEG-2 as *compressing* standard. But as techniques evolve, traditional waveform, sample-based and frame-based signals become unsatisfactory for a variety of new applications and requirements. For example, with better capturing, communication and storage techniques, the amount of digital media data from various sources is increasing dramatically that overwhelms the management capacity of traditional databases. The users of image databases prefer to find a picture or video clip that is "similar" to a given one, as compared to the traditional key words based searching. Internet surfers would like to go directly to news stories that are most interesting to them rather than to download and view the whole news video from beginning to the end. In addition, as the development of computer animation, virtual reality and video based interactive game, the boundary between computer graphics and traditional media forms such as video and audio is becoming less crucial. Both the research community and the industry come to realize that it is necessary and important to develop comprehensive systems and standards that handle various media forms:

video, audio, text and computer graphics in a consistent way. All these requirements boil down to a new concept of *content-based* multimedia system, which is also where the topic of this dissertation resides.

1.2. Content-Based Multimedia Systems

In the available media forms, text and computer graphics are synthesized from semantics with well-defined content information, while video and audio are captured and digitized signals with no clear content definition. In a content-based multimedia system, low-level signals such as audio and video are associated with high-level content information, and they are coded, transmitted, stored and managed with content awareness. Because human beings are the end user of most multimedia systems, content-based system is intuitively superior to traditional low-level signal-based systems. In the literature, a number of research works have been reported on how content analysis and semantic information can help create better multimedia systems. Here we try to overview them briefly in several (not rigid) categories.

1. **Efficient video coding**. Content information can help improve coding efficiency in several ways. Typical works in this category include model-assisted video coding, model-based video coding and eigen-face based coding. Model-assisted coding was first developed by Eleftheriadis and Jacquin [28, 29]. They designed an ellipse-fitting algorithm to locate human faces in videoconferences and a region adaptive rate control algorithm was developed for subband and H.261 encoders. Their basic idea is that more important regions (such as face region) receive more encoding bits as compared with less important regions (such as backgrounds). This approach offers an interesting *content-based* extension for traditional signal compressing algorithms. In model-based coding [3, 58, 57], a generic mesh model is designed for human faces and this model

is fitted to actual faces in time of coding. The face motion is then tracked and projected into a feature space spanned by a set of "action units" [58]. This approach is demonstrated to be able to transmit face images at the rate of several kilo-bits per second with good SNR quality. But its major problem is that it is hard to find the fitting and the tracking parameters for individual faces. Similarly, Moghaddam and Pentland [79] proposed to encode human faces with eigen-faces based on their research of human face recognition. Eigen-faces are obtained through K-L expansion of a large training data set. Actual face vectors are projected to the eigen-faces and only the projections are coded. Their reported coding performance of about 100 bytes for faces from their Photobook database.

2. **Nonlinear video structuring for indexing and retrieval**. To manage the growing amount of video data on the web, many research works are carried out to create nonlinear indexing structures from linear video sequences. Simple clustering methods have been shown to be fairy effective for content analysis stages such as scene cut detection and key frame selection [118, 35]. Some high-level abstraction stages are also proposed. For example, Yeung [114] proposed to generate a Scene Transition Graph based on the temporal position and the chromatic similarity of key frames. In [32], Zhang designed an automatic news video analysis system. A simple template is designed to detect anchor-person scene cuts and the news video is segmented into separate story units based on anchorperson detection. The story units are further grouped into domestic news, international news, weather reports and commercials. In [82], Mukhopadhyay and Smith reported an automatic web site design system for lecture media, in which the captured lecture video, audio and power-point lecture slides are synchronized based on their contents for Internet retrieval.

3. **Embedded video hotlinks**. As a nature extension of hotlinks in HTML documents, several research works have proposed to embed hotlinks in video objects. That is, semantically meaningful video objects are defined within certain spatial and temporal regions of the video sequences. When users double click mouses within these regions, the embedded hotlinks are activated. Typical examples include IBM's HotVideo [109], Veon's HyperVideo [110] systems and MIT Media Lab's Hyper-Video system [23]. In addition, the suggested "interactive TV" [44] also works in a similar manner.

4. **Similarity-based indexing and retrieval**. Early work in this area mainly focused in signal similarities, for example, color histograms [94], texture features [93, 98], edge features [119] and motion similarities [89, 25]. Some works tried to obtain certain semantic information from video data by automatic or semi-automatic definitions of *objects*, and the indexing and retrieval work is then designed based on the object features, for example, object motion orbits [13], object shape [92, 24, 43, 47, 46], spatial and temporal relationship of objects [13], *etc.* But no efficient methods available for video object definition is the major problem for these works.

5. **Standardization efforts of MPEG-4 and MPEG-7**. In recognition of the content requirements in multimedia research, MPEG committee continues their work on MPEG-1 and MPEG-2 with the proposal and development of MPEG-4 and later MPEG-7 standards. The major concern of MPEG-4 is to combine efficient media compression algorithms with content-aware media representations. In this way, the compressed video and audio can still be managed efficiently based on their contents. MPEG-4 realizes this by the definition of *objects*. Video, audio, graphics and texts are all coded and transmitted as individual objects. The objects are then composed together at the time of

playback. Under the MPEG-4 framework, the content information, once obtained, is maintained in its basic representation. Therefore, content-based processing such as indexing, retrieval, content editing, content filtering, *etc.* are all convenient tasks. MPEG-4 is an important standard in multimedia in that it offers an object-based representation and platform in which various forms of media can be processed, transmitted and stored. But MPEG-4 does not specify the process of content analysis. For example, MPEG-4 assumes video objects are created outside of its system. Similarly, the upcoming MPEG-7 is also going to by-pass the stage of content analysis. According to the latest requirement release [49], MPEG-7 "aims to create a standard for describing the multimedia content data", which will support "the increasing number of cases where the audio-visual information is created, exchanged, retrieved and re-used by computational systems". That is, the target of MPEG-7 is to create a *description* standard that will facilitate the computation processing of multimedia data. In addition, according to [49], MPEG-7 is trying to design a representation standard for high-level visual media interpretation as compared with the object-based MPEG-4 standard.

To sum up, though we do not intend to be inclusive, it is already obvious based on the surveyed works that content-based system is the trend of multimedia research. Because a human observer is the end consumer of media information, appropriate content information helps in various aspects of media processing: coding, indexing and retrieval, media interaction, media streaming and filtering, *etc.* MPEG-4 and MPEG-7 intend to create a general framework for content systems, *i.e.*, they specify mechanisms for content representation and content description, content transmission (streaming, rate control, error control, *etc.*), content playback (rendering, composing), and content storage and management (index, retrieval). But the very

gap between the traditional waveform-based media system and the content-based system, *i.e.*, content analysis stage, is left without definition.

1.3. Multimedia Content Analysis

Multimedia content analysis is the process of obtaining content information from media signals. As the concept of multimedia itself, multimedia content analysis research is a new research area. It is related to traditional research topics such as speech recognition, image understanding and computer vision, but is not totally the same. The difference includes at least the following aspects.

First, multimedia content analysis refers to a wide range of processing results that spread the media understanding spectrum. For example, in video content analysis, scheme-cut detection and key frame selection can be done based on simple signal statistics. Image similarity could be compared based on their simple statistics such as color and motion histogram, or at the object level by comparing the shape, color similarity of the objects within the images (*e.g.*, MPEG-4), or even at the high-level of image understanding by comparing the *meaning* of the images (*e.g.*, MPEG-7). The specific choice of analysis methods is highly application dependent.

Second, in multimedia systems, waveform signals are indispensable components. The content information is used to manage and improve the waveform signals, not to replace them. For example, in the videophone case, people always would like to see the original video of their partner though they may have clear idea of the content of this video, *i.e.*, who it is. Therefore, content analysis algorithms for multimedia applications have to work with various signal qualities (due to different compression qualities) and sometimes have the issue of *compressed domain* processing, *i.e.*, to work directly on the compressed signals.

Third, the end consumer of multimedia system is human, therefore, the quality

requirement on content analysis algorithms is more flexible. In some cases, for example, content-based retrieval, the performance of content analysis algorithms is still useful at certain *percentage* of accuracy that may not be acceptable for most computer vision tasks.

Fourth, multimedia content analysis can sometimes make use of information fusion over various media forms. For example, in automatic broadcast news indexing, information from video, audio and close caption could be combined in order to get effective indexing results.

In this thesis work, the research efforts are mainly focused on video analysis at the object level, *i.e.*, object detection and segmentation. Therefore, we give a brief survey on related works, which will help create a reference framework, as well as help illustrate the problems solved in this thesis.

1.3.1 Video Object Detection and Segmentation: the Problem Formulation

In multimedia research, the concept of video object (VO) was first defined by MPEG-4 standard, as atomic units of image and video content. According to [48, 87], VOs could be a semantically meaningful unit such as a talking person, a car, *etc.*, and they could also be just a signal unit. For example, conventional rectangular imagery is handled as a special case of such objects. In this work, we use video objects to refer to only the semantic units, which are generated by video analysis.

In video analysis, VO detection and VO segmentation are two related but different concepts. To detect a VO is to find the existence of it, but not necessary to find its exact boundary and regions, while the concern of VO segmentation is to find the exact location and boundary of the object. Therefore, the quality measures for detection are miss rate and false alarm rate, while the quality measures for seg-

mentation are boundary and region accuracy [115, 69]. In some cases, segmentation may be a preliminary stage for detection, but by definition, they are two different tasks.

1.3.2 Video Object Detection and Segmentation for Multimedia Applications: the Approach

General-purpose image segmentation and object detection has been difficult problems in computer vision for quite a few years. As we have already seen in the survey on content-based systems in Section 1.2., the specific analysis requirements for multimedia applications are wide-ranging, and video analysis for multimedia systems could be designed application specific. With application domain knowledge and limitations, analysis problems become solvable with today's computation powers.

So far, video analysis for the purpose of video object segmentation and detection has been investigated in a variety of applications.

1.3.2.1 Video Object Segmentation

Automatic video object segmentation has been studied extensively by the multimedia community since the introduction of the VO concept by MPEG-4. The approaches in the literature could be classified generally into several categories:

1. **Foreground and background segmentation** algorithms assume that the foreground of the picture, *e.g.*, human beings, are moving objects while the background is static objects. Therefore, the object segmentation problem is solved by moving region detection. Early works in this area include [55, 1], several similar but improved algorithms [19, 41] were also proposed to MPEG-4 as optional VO segmentation methods. The quality of these algorithms are

good when the assumptions are true, but when foreground motion stops, the performance degrades greatly.

2. **Motion field segmentation** is another type of segmentation that is based on motion. Object regions are obtained by finding the uniform regions in dense motion fields. Typical works by Wang and Adelson [107, 106] and Swahney and Ayer [8] proposed to segment the motion fields by clustering the motion field with EM algorithm. In [18, 71], boundary models such as active contours and level sets were proposed to segment the motion fields. A common problem for motion based approaches, however, is that segmentation results are only created on individual frame bases, no tracking algorithm has been designed to generate temporally consistent video objects. In addition, motion clues can not always be associated to semantically meaningful video object definitions.

3. **Multiple image feature based segmentation algorithms** segment video objects combining multiple feature clues. Altunbasak and Tekalp [6] combined color segmentation with motion segmentation by doing motion segmentation on top of over-segmented color regions. Ohm and Ma [84] introduced a concept of cluster segmentation, in which a clustering algorithm is designed over multiple image features over a pixel basis. Different features contribute to clustering process depending on their heuristic *reliable* factors. Alatan [4] proposed to combine the results of color and motion segmentation with a "rule based region processor". Though combining multiple features is theoretically meaningful, the algorithm becomes fragile and unstable when too many factors are included. Most works in this category only report experimental results on limited samples, and there is no practical system available so far.

Despite the extensive research efforts invested on automatic video object segmentation, the results are only acceptable on an application-specific basis. Therefore, many researchers seek to solve the problem with semi-automatic approaches. For example, Chalom and Bove [12, 11] designed a semi-automatic segmentation system to identify and track objects, which they later used to create a hyperlink video [23] system. In their system, the user specifies video objects interactively on the beginning frame and the computer tracks the objects temporally based on multiple image features including color, position, motion and texture. Castagno and Kunt [10], Correia and Pereira [22] and Gu and Lee [42] also proposed similar systems that create VOs with human initialization and machine tracking. Zhong and Chang [117, 116] built up a video object database consisting of several hundreds of objects with the segmentation tool they designed.

1.3.2.2 Video Object Detection

Object detection has also been studied in a number of multimedia applications. Most detection algorithms reported so far in multimedia research have been human face detection and anchorperson detection, due to their pervasive appearance in multimedia videos and their well-defined structures. As discussed in Seciton 1.2., detection of faces and anchorpersons help in efficient indexing and retrieval design in multimedia databases.

In the literature, face detection algorithms can be classified generally into two groups: color based approaches and texture based approaches. Color based works try to model human skin color in various chromatic spaces (RGB, YCbCr, HSV) with various statistical models. In [85] Oliver and Pentland modeled the face color and background color as a mixture of Gaussian distributions with Expectation-Maximization (EM) algorithm. Garcia and Tziritas [39] proposed to represent the

skin color by approximating its distribution volume in the YCrCb space with a number of linear planes. Wang and Chang [105] applied the color modeling directly to compressed MPEG macroblocks and modeled the color detection problem as a *Bayesian minimal cast decision rule* problem. In addition to color modeling, a number of recent works seek to include additional heuristics such as texture, symmetry, region ratio, etc [5, 112, 2].

Texture based works, however, try to detect face directly from the grayscale information. Sung and Poggio [97] developed a good face detection system based on multimodal gaussian clustering at MIT AI lab. Another famous face detector was developed by Rowley and Kanade [88] at CMU, based on neural networks. They later applied this face detector to news video indexing [91]. Nefian and Hayes [83] reported use of Hidden Markov Model (HMM) for face detection. Moghaddam and Pentland [78] suggested to use eigen-space decomposition methods for face detection purpose.

In general, most texture-based methods are developed from face recognition algorithms, which need various training stages. They have good detection accuracy within the scope of their training set but the missing rate climbs high for data outside the training data set. For example, most texture-based algorithms detect only faces in frontal views with limited tilt and rotation. In contrast, color-based algorithms suffer from their high false alarm rate when the background has skin type colors. Despite the various heuristics that have been proposed, color-based approaches are still not as stable as texture-based approaches. It seems it will be a fruitful direction to combine them together, but so far, no such work has been reported yet.

As compared with face detection, less works on anchorperson detection are reported. In Zhang's pioneering work [32] on broadcast news analysis, some simple

templates are created for anchorperson frames that include one and two anchorpersons, with the size and the location of the anchorperson almost fixed. The incoming video frames are compared with these templates directly. Though the author had reported good results on experimental videos, their template design is too simple to accommodate reasonable variations in different anchorperson shots. In another recently work, Liu and Huang [62] proposed to detect anchorpersons by color-based human face detection and temporal frequency based clustering method.

1.4. Thesis Outline and Contributions

This thesis is in the area of video object detection and segmentation for multimedia applications. Three applications have been studied: real time video segmentation for head-and-shoulder type videophone service, face and anchorperson detection for video indexing and retrieval, and semi-automatic generation of video objects for MPEG-4. We developed the algorithms in two directions, *i.e.*, automatic algorithms and semi-automatic algorithms.

In automatic VO segmentation and detection research, we try to tackle the problems by constraining the application domain or the problems to be solved, so that the video objects to be detected or segmented can be represented with some well-defined templates and/or models. The detection and/or segmentation problems can then be formulated as model-based object detection problems. In our work, the term *model* is used in a very general sense. In principle, any mathematical formulation that has the abstracting power to represent a category of video objects is referred to as a *model*. However, in most cases, statistical models are the most commonly used model representations. Specifically in our work, Gaussian and mixture of Gaussian models are extensively used for various model designs, *e.g.*, texture models, color models and shape models in different applications. Two types of methods are used

to create models in our work: (1) Pure training example based modeling method. In this method, all the model parameters are obtained directly from training examples, no human heuristics are used. For example, the skin color model, the face texture model and the shape model for head-and-shoulder patterns in this work are created with this method. (2) Semantic abstraction based method. This method creates models with certain human abstraction and knowledge. For example, to segment and track the head-and-shoulder pattern, a blob model is designed in this work. That is, we believe that the head-and-shoulder pattern could be represented with two blobs in appropriate ratios and positions.

In semi-automatic video object segmentation research, our major concern is to design highly efficient system to facilitate human-and-computer cooperation. Unlike previous works in this area that have their major attention to the *automatic* aspect of the system, our system design is more focused to the *interactive* aspect, *i.e.*, we care more for efficient interface design.

The structure of this thesis is organized as follows. We begin in Chapter 2 by presenting a real time video segmentation algorithm for typical videophone and videoconference applications. In this algorithm, the foreground (a person in its head-and-shoulder pattern) and the background are modeled separately. The head-and-shoulder patterns are represented with Gaussian type, blob-based region models and a Gaussian type shape model. The algorithm makes use of online information to build and track statistical models for both the background and the foreground on the fly. This algorithm is related to the foreground and background segmentation algorithms mentioned in Section 1.3.2.1, but the domain knowledge is better embedded in our models. Due to the domain constraints used and a hierarchical processing structure design, this algorithm runs using software only in real time on an Intel Pentium 200 MHz PC. Based on this real time segmentation result, two

possible applications, *i.e.* to introduce MPEG-4 standard to real time videophone service and to make use of segmentation results to further improve conventional H.263 at the very low bit rate, are discussed.

Chapter 3 extends our head-and-shoulder statistical models from online applications to offline applications by developing an automatic anchorperson detection algorithm for broadcast news video indexing and retrieval. Anchorperson patterns are modeled as head-and-shoulder patterns similar to Chapter 2. However, in order to fit the models to candidate video frames in an offline fashion, expensive searching work has to be done. To overcome the difficulty, the detection problem is decomposed into a color-based face region detection problem and a model fitting problem, with the fitting process constrained by the face detection results. Compared with previous anchorperson detection works, our approach is more flexible than [32], and more accurate than [62].

In Chapter 4, we further investigate the problem of face detection, which we studied as a functional module for anchorperson detection in Chapter 3. Unlike previous approaches, we seek to design detection algorithms with both texture and color clues. In addition, to better process the large amount of video data in the compressed domain, we design the algorithm in the compressed DCT domain rather than in the pixel domain. Part of this work is motivated by the earlier work of Wang and Chang [105]. We extend their work by designing a texture module, which is similar to Sung's [97] work in the pixel domain. However, in order to achieve the domain transition, we discuss some fundamental issues such as DCT block effect, quantization effect and feature selection.

In Chapter 5, we design and implement an interactive system for general-purpose video object segmentation, which represents our belief in the solution of the general-purpose segmentation problems. Unlike the automatic algorithms discussed in the

previous chapters of this thesis, we seek to design computer algorithms in combination with human-computer interaction mechanism at the early design stage in order to obtain good system efficiency. A new contour interpolation algorithm is designed, which enables the user to define the contour of a video object on multiple anchor frames while the computer interpolates the missing contours of this object on every frame automatically. In addition, new user interaction models are created for the user to interact with the computer, which is fundamentally different from available simple tracking approaches in the literature [12, 10, 22, 42, 116].

Finally in Chapter 6, we conclude the thesis.

In summery, the contribution of this thesis work includes the following aspects:

1. A design and implementation of a real time segmentation system for video-conference applications, which supports MPEG-4 standard for real-time applications.

2. Developed an optimal region adaptive rate control algorithm for H.263 that combines objective and subjective quality measures. Implemented the algorithm together with the online segmentation algorithm and had the system runs in real time software only.

3. A design and implementation of a highly efficient model based anchorperson detection algorithm.

4. A systematical study of face detection problem in the compressed domain. The problems discussed include preprocessing, block quantization effect and feature selection in the compressed domain.

5. A design and implementation of an efficient interactive video object segmentation system.

6. Release of several general-purpose software packages. An image processing library in C++, as well as an image manipulation library with a Microsoft Windows GUI environment is developed.

During this thesis work, publications by the author directly related to the thesis work include [68, 64, 63, 66, 65], relevant but not directly related include [69, 70, 67].

Chapter 2

Real-time Model-Based Video Object Segmentation

2.1. Introduction

With current computer system's increasing power to transmit, process and store digital video, a multimedia system's objective is no longer just to compress video signals as two-dimensional signals for transmission or storage. Instead, it should enable functionalities such as content-based transmission, manipulation, indexing and retrieval. These new functionalities make it necessary to have new models to represent video contents, and efficient ways to create them. The emerging MPEG-4 international standard intends to address the requirements of object-based video compression and representation. However, MPEG-4 does not standardize solutions for the problem of video object (VO) creation.

Segmenting objects from image and/or video data has been under intensive research. Numerous algorithms are available in the literature. However, few can really achieve automatic VO generation. Some of them only create low level image regions without any semantic meaning, such as [90, 104, 17]; others, like [55, 1, 107, 96, 17, 19, 41], depend heavily on motion, which is valid only in quite limited cases.

As such, we believe that more practical solutions to VO segmentation problems may be developed in two directions. One is to introduce some kind of human interaction in the VO segmentation process, such as the works reported in [12, 10, 22, 42, 116]. The other is to limit the application domain and use some domain-specific knowledge to guide the segmentation. For example, motion-based segmentation algorithms are actually using the domain knowledge that assumes VOs are moving while the background is static. In [22], video analysis algorithms are classified into those for online applications and those for offline applications. Obviously, a human interaction-based approach is more appropriate for offline applications while a domain knowledge guided approach is required for online, real time applications.

Videophone is a major type of real time video service in multimedia applications and its head-and-shoulder type picture pattern also offers important constraints for exploiting domain knowledge. This is the reason why this work is focused in the direction of domain constrained, real time analysis for videophone service.

In a signal compression sense, traditional transform coding algorithms such as H.261 and H.263 are well developed for videophone service. However, with intelligent video content analysis, videophone applications can be further improved. For example, by segmenting the videophone pictures into foreground objects (head and shoulder) and background object, the encoder can use different quantization and error protection to different video objects. Also the users may have the option to compose the videophone picture according to their personal tastes, *i.e.*, they may choose different background pictures, or may add some icons or text to the picture. In addition, an encoder with semantic information may introduce content-based scalability in addition to traditional quantizer adaptive scalability by discarding less important video objects. This is especially important for applications without QoS guarantee such as Internet and wireless videophone services. Actually, the

new content and video object oriented MPEG-4 standard can be efficiently applied to videophone applications as well as other applications. However, an important problem is that a reliable and real time video analysis algorithm is necessary.

In this chapter, we propose a model-based, real time video object segmentation and tracking algorithm mainly for the videophone applications. This algorithm uses inexpensive videoconference quality CCD cameras and runs software only in real time on average PC platforms. We believe it is a reasonably good algorithm for introducing the content-based approaches into videophone applications.

The essence in this algorithm is its limited application domain or context, *i.e.*, in a typical videophone setup, we assume the foreground object is one person in a head-and-shoulder pattern and the background is relatively static. Because of these limitations, we develop a statistical model to represent appropriate *a priori* knowledge and then try to fit the model with the actual video data. More specifically, our algorithm segments the input video into three VOs: a background, a head/face and a shoulder. The domain knowledge is modeled from two aspects. One is a blob modeling of each VO region's color and spatial distribution, and the other is a shape modeling. The segmentation and tracking process is actually the process of fitting and updating of the region model and shape model. In order to generate video objects with refined boundaries, novel spatial and temporal filters are designed as extensions of the statistical modeling framework. In addition, a hierarchical structure is used at the system level to coordinate the processing and to speed up the performance.

The structure of this chapter is as follows. First in Section 2.2., we discuss blob-based region modeling and Kalman filter for blob tracking. In Section 2.3., we introduce shape modeling. In Section 2.4., a hierarchical system structure is introduced to improve system performance. Segmentation experiments are presented

in Section 2.5.. In Section 2.6., we discuss the application of the segmentation algorithm to real time videophone services. Finally the chapter is concluded in Section 2.7..

2.2. Blob Based Region Modeling and Tracking

2.2.1 Motivation and Context

This work is motivated by the widely used "chroma-key" technique and the human body tracking work of "Pfinder" [111]. In addition, it is also related to the foreground/background segmentation work by Lettera and Mastera [55]. The basic nature of the algorithm is an online one. First, assume a background scene that contains no foreground, which enables the creation of a background model. Then when the foreground enters, another model is created for the foreground. As discussed in the introduction, the purpose of modeling is to find proper representation of domain knowledge that helps with segmentation and tracking.

2.2.2 Foreground Model

In video telephony, typical foreground can be represented as a "head-and-shoulder" pattern. We model it with two connected "blob"s. Here the definition of blob is similar to that in [111], *i.e.*, each blob has a spatial (x, y) and chromatic (Y, U, V) Gaussian distribution and a support map which indicates whether a pixel is a member of a blob. In this model, each pixel is represented by a feature vector (x, y, Y, U, V). The feature vectors of the pixels belonging to blob k have a Gaussian distribution with mean vector $\mathbf{m}_k$ and covariance matrix $\mathbf{C}_k$. Because of their different semantics, the spatial and chromatic distributions are assumed independent. That is, the matrix $\mathbf{C}_k$ is assumed block-diagonal.

In addition, we introduce some definitions related to blob modeling as follows. First, the support map $s_k(x, y)$ for blob k is defined as

$$s_k(x,y) = \begin{cases} 1 & \text{if pixel (x,y) belongs to blob k, k=1, 2, 3, } \ldots, \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

Based on $s_k(x, y)$, blob k's *containing rectangle* rect_k is defined as the rectangle with the following coordinates:

$$\text{rect}_k = \text{rectangle}[(x_t, y_t), (x_t, y_b), (x_b, y_t), (x_b, y_b)], \tag{2.2}$$

where

$$\begin{cases} x_t = & \sup_{s_k(x,y)=1}(x) \\ x_b = & \inf_{s_k(x,y)=1}(x) \end{cases} \text{, and } \begin{cases} y_t = & \sup_{s_k(x,y)=1}(y) \\ y_b = & \inf_{s_k(x,y)=1}(y) \end{cases} . \tag{2.3}$$

For segmentation purposes we also define a cumulative support map $s(x, y)$ for each image as:

$$s(x,y) = \begin{cases} k & \text{if } s_k(x,y) = 1, \text{ k=1, 2, 3, } \ldots, \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

In addition, we define the entire set of the support maps of foreground blobs as the foreground map $f(x, y)$:

$$f(x,y) = \begin{cases} 1 & s(x,y) \neq 0, \\ 0 & s(x,y) = 0. \end{cases} \tag{2.5}$$

The relation between these concepts can be seen in Figure 2-1: the left image illustrates two blobs, the middle image shows a support map with containing rectangles, and the right image is a foreground map.

The rationale behind blob modeling is that it represents an image region that

has chromatic and spatial similarity. By the definition of a blob and its associated support map, low-level pixel oriented segmentation problem is associated with high-level semantically meaningful blob tracking. In this way, high-level *a priori* knowledge can be used to guide pixel segmentation.

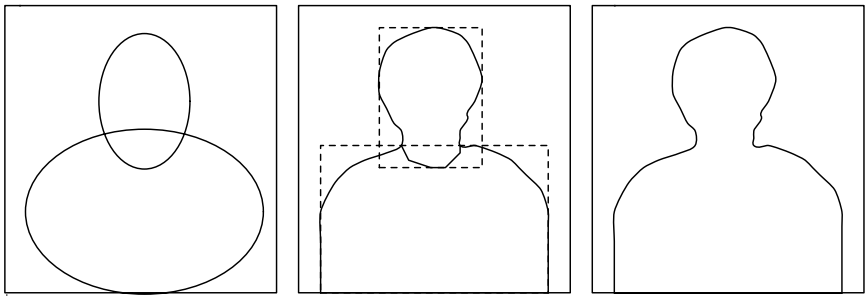


Figure 2-1: Left: blob representation, middle: support map with containing rectangles, right: foreground map.

2.2.3 Background Model

The background is modeled as a texture map that varies over time. In common videophone applications, we assume the camera is static and there are no fast and major background changes. In this context, there are still several sources that may introduce temporal color variations in the background pixels and thus influence an accurate modeling. They are enumerated as follows.

1. The thermal noise of the camera sensor. This is mainly influenced by the quality of the camera. Two factors: noise stability and magnitude are related to our modeling work.

2. The gradual changing of background for some reason. For example a foreground human may move some items in the background or there may be changes in illumination, *etc.*

3. The AGC (automatic gain control) effect of the camera. This is quite obvious when the foreground moves into or out of the scene, or closer or farther away from the camera and changes the exposure of the camera. In these cases, the AGC mechanism introduces noticeable luminance variation in the background pixels even though the background itself does not change at all. Fig. 2-2 and Fig. 2-3 show our test results using Intel's Proshare videoconference camera, which does not have an AGC switcher. Fig. 2-2 is the histogram of the inter-frame variation of the luminance Y for background pixels when there is no foreground in the scene. Fig. 2-3 is the inter-frame Y variation of the same background pixels (not occluded by foreground) when a person enters the scene three meters away from the camera. We can see that the shift can be decomposed into two parts: a uniform shift and a pixel dependent shift[1].

4. The shading effect brought about by the foreground motions. This factor can be compensated in part by introducing the normalized chromatic vector $(U^* = U/Y, V^* = V/Y)$.

Given all these factors, we model each pixel in the background as a Gaussian distribution in vector space (U^*, V^*) with mean vector $\mathbf{m}_0$ and covariance matrix $\mathbf{C}_0$. In the segmentation loop, the background model is created and updated as follows. First the uniform shifting vector $\mathbf{diff}_t$ is estimated and compensated:

$$\mathbf{diff}_t = \frac{\sum_{f(x,y)=1}(\hat{\mathbf{y}}_t(x,y) - \hat{\mathbf{y}}_{(t-1)}(x,y))}{\sum_{(x,y)\in\mathcal{I}} f(x,y)}, \tag{2.6}$$

[1]According to our experiments, this effect is less important for those cameras with an AGC switch that can disable the AGC function. We have tested Sony's SSC S20 CCD camera with AGC function turned off and found that the foreground induced background color shift can be ignored in most cases.

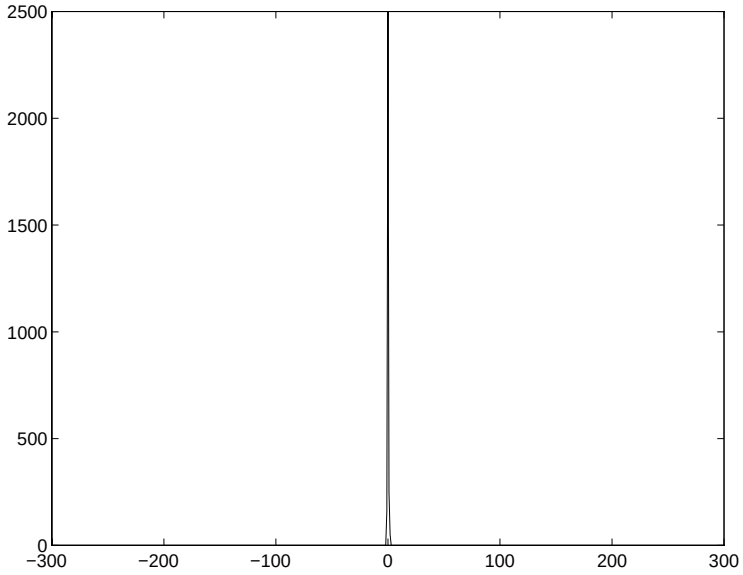


Figure 2-2: AGC effects illustration. When there is no foreground in the scene, the histogram of visible background pixels' inter-frame luminance (Y) variance distribution is approximately a narrow Gaussian peak centered at zero. This is mainly caused by the thermal noise of CCD camera.

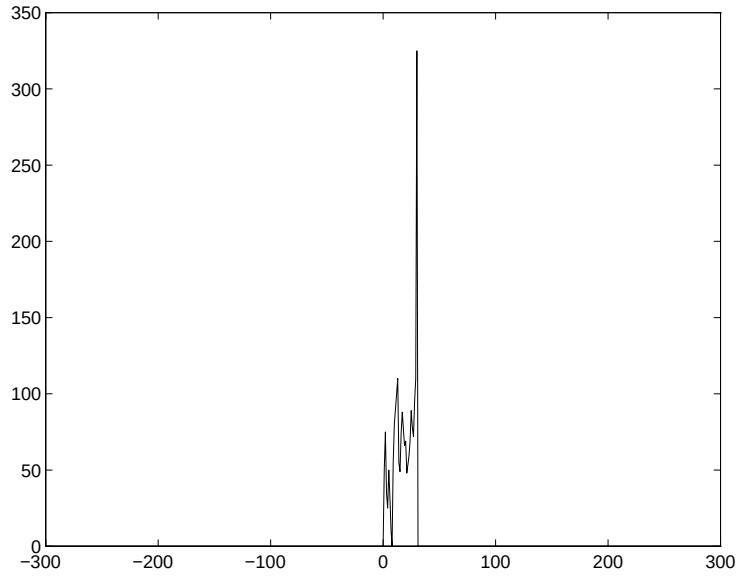


Figure 2-3: AGC effects illustration. When some foreground enters the scene, the histogram of visible background pixels' inter-frame luminance (Y) variance distribution can be interpreted as including two components: a uniform shifting part (corresponding to the highest peak in the right side) and a region adaptive part.

$$\mathbf{m}_{0,t}(x,y) \quad = \quad \mathbf{diff}_t + \mathbf{m}_{0,t-1}(x,y), (x,y) \in \mathcal{I}, \tag{2.7}$$

where $\hat{\mathbf{y}}_t(x,y)$ and $\mathbf{m}_{0,t}(x,y)$ are the feature vector and model parameters for the current background pixel at the spatial position (x,y) and temporal position t (in succeeding references, $\hat{\mathbf{y}}_t$, $\mathbf{m}_{0,t}$ or $\hat{\mathbf{y}}$, $\mathbf{m}_0$ may be used when (x,y) and/or t information are not important), and vector $\mathbf{diff}_t$ is a frame level uniform shifting factor. As indicated in Eq. (2.6), $\mathbf{diff}_t$ is obtained by averaging the temporal feature vector difference over those background pixels that are not occluded by the foreground ($f(x,y) = 0$). However, the compensation in Eq. (2.7) updates every pixel within the background model, including those pixels occluded by the foreground as well (in Eqs. (2.6), (2.7) $\mathcal{I}$ refers to the pixel set in one frame). This accounts for the uniform shifting effect of the above mentioned third factor and is useful to model those uncovered background pixels.

In addition, each visible background pixel has its statistical parameters updated as:

$$\mathbf{m}_{0,t} = \alpha * \hat{\mathbf{y}}_t + (1-\alpha) * \mathbf{m}_{0,t-1} \ , \ (0 \leq \alpha \leq 1). \tag{2.8}$$

This compensates for the above mentioned second, third and fourth factors.

Note that unlike the foreground model, each pixel of background is modeled individually. Or to express in a uniform way, the feature vectors for the background model can also be put into the vector space (x, y, U^*, V^*) by implicitly including the spatial coordinate of each pixel (x,y). This approach can accommodate a variety of complex backgrounds without limiting them to fit to a structure like head-and-shoulders foreground.

2.2.4 Region Classification

With the available foreground and background statistical models, it is straightforward to classify pixels into different regions by their statistical likelihood. In our work, MAP (*maximum a posteriori probability*) principle is used for the classification. We have two foreground classes (*shoulder* and *head*, $k = 1, 2$) and one background class ($k = 0$). To compensate the shading effect, we choose to use feature vector $\hat{\mathbf{y}} = (x, y, U^*, V^*)$ instead of (x, y, Y, U, V) for foreground blobs as well (the major modeling principle remains the same). So the log likelihood is expressed as:

$$\ln(p(\hat{\mathbf{y}}|\Omega_k)) = -(\hat{\mathbf{y}} - \mathbf{m}_k)^T \mathbf{C}_k^{-1} (\hat{\mathbf{y}} - \mathbf{m}_k) - \ln(\det(\mathbf{C}_k)), \ (k = 0, 1, 2), \tag{2.9}$$

where Ω_k represents the event that the pixel belongs to class k. Based on MAP, each pixel is labeled in the support map as:

$$s(x, y) = \arg\max_k(\ln(p(\Omega_k|\hat{\mathbf{y}}))) = \arg\max_k[\ln(p(\hat{\mathbf{y}}|\Omega_k) + \ln(p(\Omega_k))], \tag{2.10}$$

where $\ln(p(\Omega_k)))$ is estimated based on typical videophone pictures.

To convert the classified pixels into meaningful regions, two steps come next. First the foreground pixels are processed with morphological filters to create a simple connected foreground map $f(x, y)$. Second the support map $s(x, y)$ is obtained by blob growing, *i.e.*, each blob is grown out within the foreground map from their blob center to create a simple connected support map. This is illustrated in Figure 2-4.

2.2.5 Blob Tracking Procedure

In this section, we discuss the basic tracking procedure in two loops: model initialization loop and tracking loop.

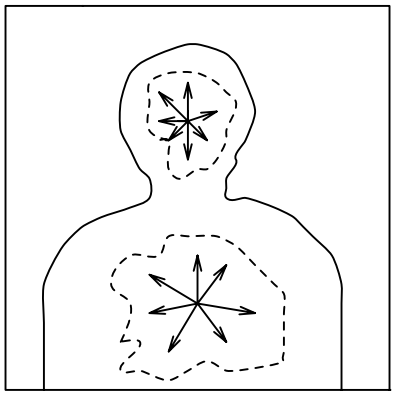


Figure 2-4: Blob region growing illustration. Each blob's support map $s_k(x, y)$ is grown out from their blob centers on top of the foreground map $f(x, y)$.

2.2.5.1 Initialization Loop

The purpose of initialization loop is to detect "head-and-shoulder" type foreground and to create the foreground and background models. Its logic steps are illustrated in Fig. 2-5. At the beginning, the background only scene is captured and a background model is created. When a foreground enters, the system detects model deviation and tries to analyze the size, speed and shape of the possible foreground and judge the likelihood of being a head-and-shoulder foreground. When a valid foreground is detected, a foreground model is created and the system enters the tracking loop.

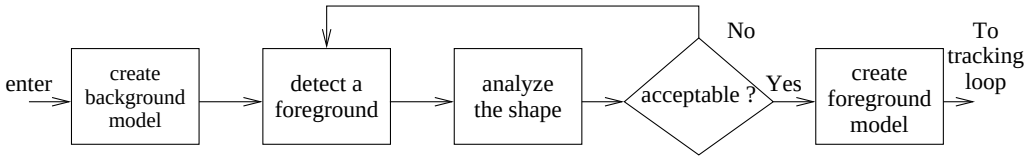


Figure 2-5: Flowchart of the initialization loop.

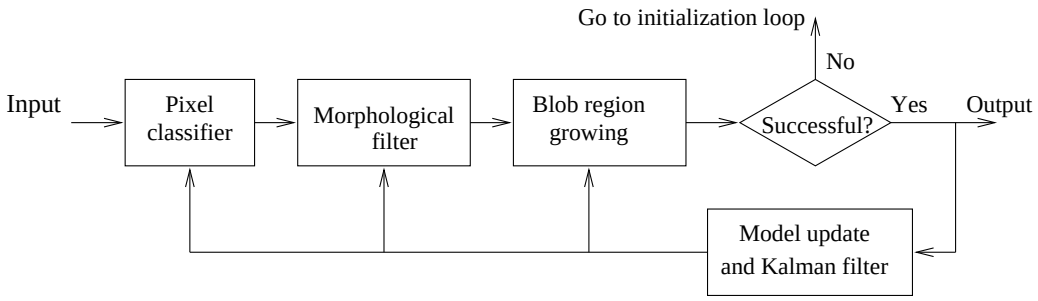


Figure 2-6: Flowchart of the basic tracking loop procedure.

2.2.5.2 Tracking Loop

The flowchart of tracking loop is shown in Fig. 2-6. As discussed in Section 2.2.4, the major steps of region segmentation are pixel classification, foreground morphological filtering and blob region growing. At the model-level, blobs are tracked with Kalman filter. Unlike low-level pixel classification, blob tracking is carried out at the model-level with semantic meanings. Though in this work, the major concern of segmentation is to get an accurate support map for each blob rather than to track the semantic information of blob motion, still good modeling and tracking of blobs are important because the tracked blobs carry the statistical model parameters, which are important for support map segmentation.

In this work, each blob is tracked independently with Kalman filter. The observation vector for blob k includes the blob's center (x_k, y_k) and blob's statistical width and height (w_k, h_k):

$$\hat{\mathbf{Y}}_k = (x_k, y_k, w_k, h_k). \tag{2.11}$$

Here w_k and h_k are defined as $w_k = 2\sigma_{x_k}$ and $h_k = 2\sigma_{y_k}$. They are obtained from the feature vector covariance matrix $\mathbf{C}_k$. The dynamic model is a discrete Newtonian physical model of rigid body motion, which has the form:

$$\hat{\mathbf{X}}(t + \Delta t) = \mathbf{\Psi}(\Delta t)\hat{\mathbf{X}}(t) + \xi(t), \tag{2.12}$$

where $\hat{\mathbf{X}}$ is the state vector, $\mathbf{\Psi}$ is the state transition matrix and ξ is the noise term. The $12D$ state vector $\hat{\mathbf{X}}$ and noise vector ξ contain four variables for the position of observation vector $\hat{\mathbf{Y}}$, four for the velocity and four for the acceleration, *i.e.*,

$$\hat{\mathbf{X}}(t) = \begin{pmatrix} \hat{Y} \\ V \\ A \end{pmatrix}, \text{ and } \xi(t) = \begin{pmatrix} \xi_{\hat{Y}} \\ \xi_V \\ \xi_A \end{pmatrix}. \tag{2.13}$$

From Newtonian physics, we have

$$\mathbf{\Psi}(\Delta t) = \begin{pmatrix} I & I\Delta t & 0 \\ 0 & I & I(\Delta t)^2 \\ 0 & 0 & I \end{pmatrix}. \tag{2.14}$$

In practice, the Kalman filter is used to predict the model parameters of each blob in the next frame, which is the start point of region classification discussed in Section 2.2.4. In return, the result of region classification in current frame is used to update the blob model parameters $\mathbf{m}_k$, $\mathbf{C}_k$, $(k = 1, 2)$, background model parameters $\mathbf{m}_0$, $\mathbf{C}_0$. It is also the observation input driving the Kalman filter for the next prediction.

Sometimes when the foreground moves too fast for the filter to follow or just moves out of the scene, the system can not find appropriate support maps at the predicted positions of the current frame. In these cases, as indicated in Fig. 2-6, the system automatically changes its status back to the initialization loop.

2.3. Shape Modeling and Recognition

2.3.1 Shape Analysis' Role for Segmentation and Tracking

In the previous sections, a blob based region modeling and tracking approach was discussed. Blobs incorporate domain knowledge quite naturally into the segmentation problem. However, this approach is mainly based on the chromatic cues, which makes its performance highly dependent on the camera's quality and the color contrast between the foreground and background. For real time tracking purposes, it is also likely that errors accumulate during the course. In order to stabilize the blob tracking process, the system should be adjusted from time to time. In our work, shape cues are used to meet the requirement.

A basic intuition to support this idea is that people can sometimes identify the head region and the shoulder region only based on the object silhouette. However, because the foreground is always in motion and its silhouette keeps changing, it is not in every case that we can analyze the silhouette successfully for segmentation purpose. For example, in Fig. 2-7 we have two foreground shapes to be analyzed. In the left image, the shape feature is strong and the head region and shoulder region can be easily separated based on the foreground shape information, while in the right image, the shape feature is not so obvious. Therefore, shape analysis only works as an adjustment or auxiliary approach to previous region based approach.

To solve this problem, a *shape recognition* module is added into the shape analysis procedure. We define those shapes with strong shape features as *canonical head-and-shoulder shape*. Those shapes outside this category are named *non-canonical shape*. If a foreground shape belongs to the canonical category, we are sure that we can locate the head region and shoulder region with a high reliability only based on shape information, then the head blob and shoulder blob are located with shape cues.

Figure 2-7: Comparison of the limits of shape analysis for blob tracking. In the left image, the shape feature is strong and the head region and the shoulder regions can be easily separated based on the foreground shape information. But in the right image, the shape feature is not so obvious and shape analysis is sensitive to noise.

Otherwise the blobs are tracked with chromatic cues as discussed in Section 2.2.. In this way, we can get the most out of the domain knowledge to stabilize the tracking algorithm.

2.3.2 Shape Modeling

Though a quantitative definition of canonical or non-canonical category is hard to give and may depend on specific algorithm of shape analysis, inclusiveness is not the requirement of this work. Rather, because the purpose of shape analysis is to stabilize the region-based blob tracking, we can define the canonical shape category as a limited size. That is, we do not intend to include all the shapes that can be used to segment the foreground into head and shoulder regions with certain simple shape-analysis algorithm. Instead, only those with strong shape features are included in the category to make sure that if a shape is accepted as a canonical one, then the segmentation output based on the shape analysis is highly reliable. In this sense, we model the canonical shape category as a high dimensional Gaussian distribution and create the model by statistical learning.

Fast Vectorization

First, a vectorization algorithm is used to convert a shape into a feature vector. Many algorithms are available in the literature to do this task, such as chain code, Fourier descriptor, polygon approximation, B-spline, *etc.* In this work, a fast algorithm is designed to fit in with real time applications. It is illustrated in Fig. 2-8: the foreground region is divided uniformly into N stripes in the vertical direction and the horizontal center and the width of each stripe are measured to constitute a $2N$ dimension feature vector $\mathbf{v}$.

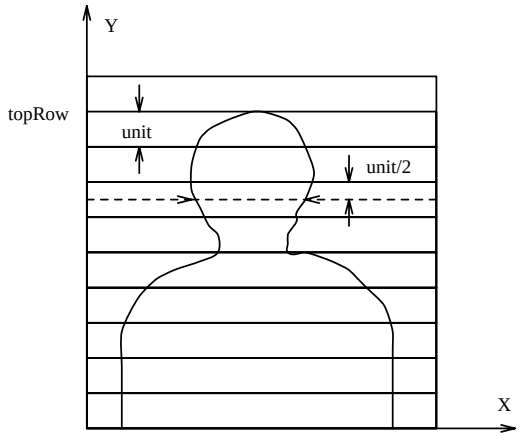


Figure 2-8: Stripe based shape vectorization illustration. Foreground is divided uniformly into N stripes in the vertical direction. For each stripe, its horizontal center and width are used as components for a feature vector. In this way, a $2N$ dimensional feature vector is used to represent the foreground shape.

Suppose the input of the algorithm is a segmented foreground picture f with each background pixel $f(x, y) = 0$ and foreground pixel $f(x, y) = 1$. The origin of the image coordinates is at the lower left corner of the image. Then the detailed algorithm may be expressed with the following pseudo-code.

```
(1)   topRow=sup_{f(x,y)=1}(y);
(2)   unit=(topRow)/N;
```

```
(3)   k=0;
(4)   for( row=unit/2; row<topRow; row+=unit ) {
(5)      start(k)=inf_{f(x,row)=1}(x);
(6)      end(k)=sup_{f(x,row)=1}(x);
(7)      k++;
(8)   }
(9)   center0=0;
(10)   for( i=0; i<N; i++ ) {
(11)      width(i)=end(i)-start(i);
(12)      center(i)=(end(i)+start(i))/2;
(13)      center0+=center(i);
(14)   }
(15)   center0 /= N;
(16)   k=0;
(17)   for( i=0; i<N; i++ ){
(18)      v(k++)=width(i)/unit;
(19)      v(k++)=(center(i)-center0)/unit;
(20)   }.
```

In above pseudo-code, (1)-(2) find the width (unit) of sampling stripes, (3)-(8) actually do the sampling, (9)-(14) convert the sampled position data (start(i) and end(i)) into width and center data, finally in (15)-(20), the sampled data is further normalized with respect to size (dividing by unit) and position (center(i) being measured with respect to center0, which is obtained by averaging the sampled data), producing the feature vector $v(i)$, which represents only the shape information.

This algorithm is actually a controlled polygon approximation of the original shape. The Euclid distance of vector $\mathbf{v}$ is a good indication of the shape difference

because of the polygon approximation nature. Though the shape vector is not rotational invariant, for most real time videophone applications it is not necessary to recognize rotational invariant shapes. The accuracy of the approximation depends on the choice of N and the complexity of foreground shape. According to our observation, $N = 15$ is big enough for most videophone applications in QCIF size.

Gaussian Modeling

With above vectorization algorithm, the canonical shape category Ω is modeled as a $2N$ dimension unimodal Gaussian distribution. The distribution can be characterized by a mean vector $\bar{\mathbf{v}}$ and a covariance matrix Σ. The likelihood of a shape vector $\mathbf{v} = \bar{\mathbf{v}} + \tilde{\mathbf{v}}$ is given by:

$$P(\mathbf{v}|\Omega) = \frac{\exp(-\frac{1}{2}\tilde{\mathbf{v}}^T\Sigma^{-1}\tilde{\mathbf{v}})}{(2\pi)^N \det(\Sigma)^{1/2}}. \tag{2.15}$$

A sufficient statistic for characterizing the likelihood is the Mahalanobis distance:

$$D = \tilde{\mathbf{v}}^T\Sigma^{-1}\tilde{\mathbf{v}}. \tag{2.16}$$

In practice, the mean $\bar{\mathbf{v}}$ and covariance Σ are obtained through a set of training shapes.

2.3.3 Eigen-Analysis and Classification

In Eqs. (2.15), (2.16), the covariance matrix Σ is $2N$ by $2N$. In order to reduce the computation complexity, the matrix is decomposed via an eigenvector transform:

$$\Sigma = \Phi^T\Lambda\Phi, \tag{2.17}$$

where Φ is the eigenvector matrix and Λ is the corresponding diagonal matrix of eigenvalues. With eigenvector transform, the likelihood equation becomes:

$$P(\mathbf{v}|\Omega) = \frac{\exp(-\frac{1}{2}\sum_{i=1}^{2N}\frac{b_i^2}{\lambda_i})}{(2\pi)^N \det(\Sigma)^{1/2}}, \tag{2.18}$$

where $\mathbf{b} = \mathbf{\Phi}^T\tilde{\mathbf{v}}$ is the new vector under the orthogonal transform. Similarly, the Mahalanobis distance is converted to:

$$D = \sum_{i=1}^{2N}\frac{b_i^2}{\lambda_i}. \tag{2.19}$$

In principle, eigenvectors correspond to the principal axes of the sub-vector space and the eigenvalues are the corresponding principal variance. Although in above orthogonal transform, all the $2N$ eigenvectors are necessary to represent the distribution exactly, only a small number K ($K \ll 2N$) of them are generally needed to encode the samples within the subspace with tolerable errors. These K vectors are often called principle components and the approach called principle component analysis (PCA). With PCA, for each vector $\mathbf{v}$, only the first K projects of b_i are necessary for the computation of likelihood in the Eq. (2.18) or Mahalanobis distance in Eq. (2.19). The computation is significantly reduced.

In practice, the thresholding of the Mahalanobis distance Eq. (2.19) is used in our work and it is approximated with two thresholding inequalities:

$$D_1 = \sum_{i=1}^{K} b_i^2/\lambda_i < T_1, \tag{2.20}$$

$$D_2 = ||\mathbf{v}||^2 - \sum_{i=1}^{K} b_i^2 < T_2. \tag{2.21}$$

A shape is determined as in the canonical shape category only if its feature vector meets above two inequalities. Here D_1 is the Mahalanobis distance and D_2 is the

energy of the feature vector's projection on the complementary space of the subspace spanned by these first K eigenvectors. In other words, we require that the projection inside the subspace is close to the center, and the projection energy outside the subspace is small. This thresholding by projection may incur some error in the mathematical sense. However, the classification error is tolerable as compared with computation benefits.

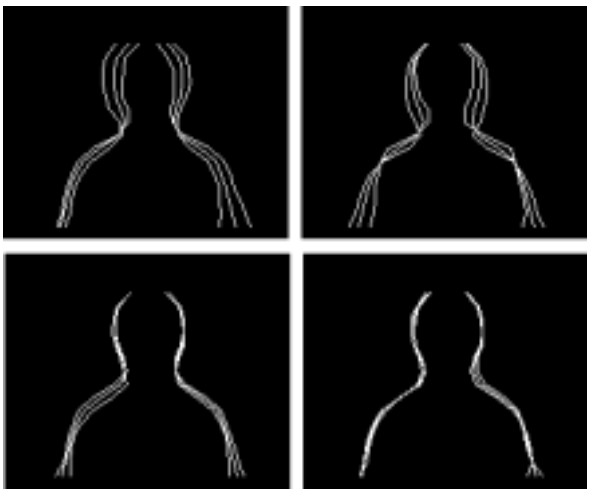


Figure 2-9: Canonical shape category's eigen-shape illustration. From left to right, top to bottom, four subfigures correspond to the first four eigen-shapes of canonical shape category. In each subfigure, three shapes are overlaid that correspond to three vectors: $\bar{\mathbf{v}}, \bar{\mathbf{v}} + a\phi_i, \bar{\mathbf{v}} - a\phi_i$, where a is a weighting factor and ϕ_i is the ith eigenvector.

In the training process, about 200 pictures of canonical shape category[2] were used. The result of eigen-analysis shows that the first six eigenvectors cover 92 percent of total energy. This is the thresholding K we used for our experiment. The shapes corresponding to the first four eigenvectors that obtained from our training

[2]Training samples are chosen by human observation, but their statistical model is created by the computer. This is the process of machine learning. In our experiment, training samples were obtained from several persons. For wider application domain, more training samples should be obtained or users may train the machine with each person's own training samples.

set are shown in Fig. 2-9. They are ordered from left to right, top to bottom. In each figure, three shapes are overlaid that correspond to three vectors: $\bar{\mathbf{v}}, \bar{\mathbf{v}}+a\phi_i, \bar{\mathbf{v}}-a\phi_i$, where a is a weighting factor and ϕ_i is the i-th eigenvector. Readers can observe the distribution of shape changing along the first several principal eigenvectors and have a general sense of the canonical shape category's shape distribution discussed in this work.

2.3.4 Shape Assisted Blob Tracking

If a foreground shape is a canonical shape, by definition its shape features is used to segment the foreground into head region and shoulder regions, or in other words, to segment the foreground map $f(x, y)$ into a support map $s(x, y)$ of head blob and shoulder blob. A *containing rectangle* based algorithm is designed to segment a foreground map $f(x, y)$ into two blobs by setting a vertical threshold t:

$$s_1(x,y) = \begin{cases} f(x,y) & \text{if } y > t\ , \\ 0 & \text{otherwise.} \end{cases} \tag{2.22}$$

and

$$s_2(x,y) = \begin{cases} f(x,y) & \text{if } y < t, \\ 0 & \text{otherwise.} \end{cases} \tag{2.23}$$

If we use operator $\mathcal{S}(s_k(t))$ and $\mathcal{S}(\text{rect}_k(t))$ to represent the size of support map k and size of its containing rectangle at the thresholding t respectively, *i.e.*,

$$\mathcal{S}(s_k) = \sum_{(x,y)\in\mathcal{I}} s_k(x,y), \tag{2.24}$$

$$\mathcal{S}(\text{rect}_k) = (y_t - y_b)(x_t - x_b), \tag{2.25}$$

where $\mathcal{I}$ represents the image pixel set, y_t, y_b, x_t, x_b are defined in Eqs. (2.2), (2.3), then the final segmentation thresholding T is chosen as

$$T = \arg\min_t \Big\{ \sum_{k=1}^{2} [\mathcal{S}(\mathrm{rect}_k(t)) - \mathcal{S}(s_k(t))] \Big\}. \tag{2.26}$$

This algorithm is illustrated in Fig. 2-10. Note the coordinate system and the horizontal thresholding line that segments the foreground into head and shoulder regions. The purpose of this algorithm is to approximate the foreground map with two containing rectangles and choose the segmentation that minimizes the approximation error.

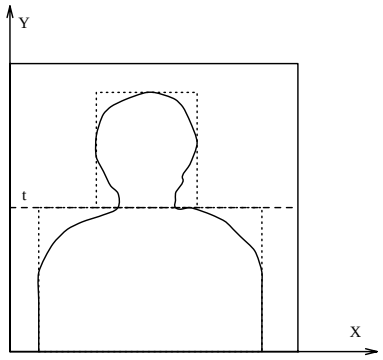


Figure 2-10: Illustration of shape-assisted blob region location. The foreground map $f(x, y)$ is approximated with two containing rectangles. The thresholding value t is chosen to minimize the approximation error.

2.4. Hierarchical System Design

Though the region and shape statistical models reduce much analysis complexity, in practice, we find that to update the blob model parameters frame by frame still involves expensive computation. In addition, because of the statistical nature of this segmentation algorithm, noise is inevitable and further filtering is necessary to improve the quality. In order to solve these problems while limiting the overall

computation complexity of the algorithm, a hierarchical structure is designed at the system level.

2.4.1 Hierarchical Architecture

In the new hierarchical design, the tracking loop of Fig. 2-6 is updated into Fig. 2-11, *i.e.*, an input image is first sub-sampled by M in both horizontal and vertical directions. Model analysis (both region based blob model and shape model) and tracking are carried out in the obtained lower resolution image. The processing result is then up-sampled and further refined in the original resolution to produce the final output.

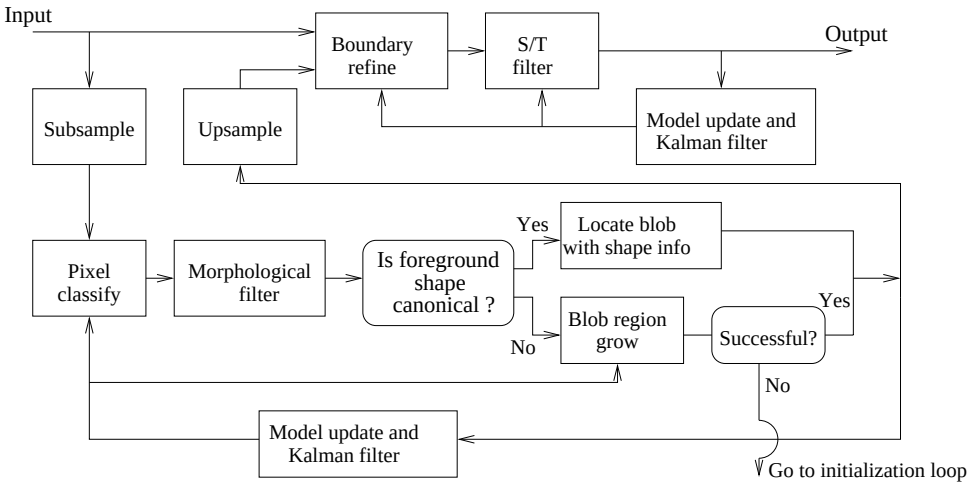


Figure 2-11: Flowchart of the hierarchical tracking system.

The benefits of this structure come from two aspects. First, because the statistical models are tracked and updated in the lower resolution image, the computation complexity is reduced by M^2. Second, when the segmentation result in the lower resolution image is mapped back to the full resolution image, only the boundary

blocks[3] are further processed by the spatial and temporal filters that are designed to suppress noise and improve the boundary quality. All the interior blocks are skipped. One drawback, however, is that two versions of the background model are to be maintained, one in the lower and the other in the full resolution (for foreground model we can maintain just one set of model parameters and convert them between different image resolutions). But compared with the benefits, this is not a big problem.

2.4.2 Processing on Sub-sampled Image

In Fig. 2-11, most function modules for lower resolution image processing were discussed in previous sections. The shape modules work as anchors for the region based tracking modules. They run in loop to find canonical shapes and locate the position of each blob. If the shape-based approach fails, the region-based blob analysis maintains its tracking with blob region growing module and information from Kalman filter. At this stage, if system still can not find the expected blobs at the predicted positions, it changes its status back to the initialization loop. Unlike Fig. 2-6, here in Fig. 2-11, because of the incorporation of the shape analysis modules, the tracking procedure is more resistant to noise in chromatic information and thus can avoid error accumulation in the tracking course.

2.4.3 Refining Processing on Full Resolution Image

In the full resolution layer, a boundary-refining module and a joint spatial and temporal (S/T) filtering module are designed to improve the boundary quality. In the three VOs we are going to get: background, head, and shoulder, head plus shoulder together is the foreground, only the boundary between the foreground and

[3]One pixel in lower resolution image is mapped into one M by M block in full resolution image.

background are considered. If a head VO or a shoulder VO is required individually, the boundary between them is approximated with their containing rectangle as defined in Eq. (2.2).

In the hierarchical structure, each low-resolution-image pixel maps to a M by M block in the full resolution image. Both boundary-refining module and S/T filtering module process only the boundary block pixels.

Boundary refining module processes image in one frame to improve the spatial smoothness of final boundary. It includes three steps. First, pixels in the boundary blocks are classified based on the foreground model and background model in the full resolution layer. Second, morphological filters are used to connect the segmentation results in each boundary block with interior block regions, so as to produce a simple connected foreground map. After that, a relaxation procedure is carried out to improve the smoothness of the foreground boundary. The relaxation process examines each boundary pixel in the foreground map $f(x, y)$, and see if it should be flipped so as to improve the boundary smoothness. This is converted to a statistical decision problem as follows.

Let $\Omega_k, (k = 0, 1)$ represents the events $f(x, y) = k$, $(k = 0, 1)$. For each boundary pixel, its MAP classification equation is

$$\ln(p(\Omega_k|\hat{\mathbf{y}})) = \ln(p(\hat{\mathbf{y}}|\Omega_k)) + \ln(p(\Omega_k)). \tag{2.27}$$

The first term in the right side can be obtained from Eq. (2.9). The second term $\ln(p(\Omega_k))$ works as a *smoothness measure*, which represents *a priori* knowledge. Because smoothness is a spatial feature, we define the *smoothness measure* of a boundary locally for each of its boundary pixel as [1] does. A priori density $p(\Omega_k)$

is modeled by a Markov random field considering a 3×3 neighborhood:

$$p(\Omega_k) = \frac{1}{Z} \exp\{-E(\Omega_k)\}, \ (k = 0, 1), \tag{2.28}$$

where Z is a normalizing factor and the energy term E is defined as

$$E(\Omega_k) = (n_B(k)B + n_C(k)C). \tag{2.29}$$

In Eq. (2.29), $n_B(k)$ and $n_C(k)$ are the homogeneity measure of the neighborhood if current boundary pixel is labeled as k. They are obtained as follows. Each current boundary pixel constitutes eight *pixel-pair*s with its eight neighboring pixels. If both pixels in a pixel-pair have the same label, this pixel-pair is a *homogeneous pair*, otherwise it is an *heterogeneous pair*. n_B is the number of those heterogeneous pairs that are in vertical or horizontal positions and n_C is the number of those in diagonal positions. B and C are two weighting factors that represent the distance factor of those pixel-pairs in different positions in relation to the boundary pixel under consideration. We have $B = \sqrt{2}C$. The 3×3 neighborhood used for boundary smoothness measure is illustrated in Fig. 2-12.

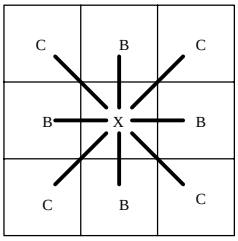


Figure 2-12: Illustration of the 3×3 neighborhood used for boundary smoothness measurement in the boundary relaxation procedure. 'X' is the current boundary pixel. Eight thick bars represent eight pixel pairs.

After the spatial boundary refining, a seven-point 3D spatial-temporal median

filter is used on the foreground map:

$$\text{med}_t(x,y) = \text{med}_7[I_{t-1}(x,y), I_{t+1}(x,y), I_t(x-1,y), I_t(x,y), I_t(x+1,y), I_t(x,y-1), I_t(x,y+1)]. \tag{2.30}$$

The purpose of this median filter is to suppress the temporal high frequency noise on the boundary, which will be quite annoying when the segmented VOs are played back with an MPEG-4 player. Notice that this filter introduces delaying time of one frame. For real time applications, higher order median filter is not desirable.

2.5. Segmentation Experiments

The algorithm is implemented on PC platforms with a variety of video capture hardware, including Intel's Proshare videoconference Kit, Intel's Create&Share Camera Pack and Sony's CCD SSC-S20 camera with Intel's Brooktree capture card. The segmentation performance is 15 fps (frames per second) on a Pentium-200 for QCIF (176×144) size input videos.

Due to the online feature of the algorithm, we could not use standard sequences in the test. Instead, several testing sequences are captured for testing purpose. One of them, in QCIF size, YUV format, 800 frames in length and 10 fps is available on the website[4]. This video sequence contains one person in his head-and-shoulder pattern as foreground with moderate motion.

Fig. 2-13 shows the segmentation result on one frame (the 500th frame) of the testing video sequence. Fig. 2-13(a) is the original input frame, (b) is the segmented foreground VO and (c) is the segmented head VO. Note that the boundary between the head VO and shoulder VO is approximated with their containing rectangle boundaries. In addition, though the capacity of our segmentation algorithm is to

[4] `http://www.ctr.colulmbia.edu/~luoht/research/rvSeg`

segment the input frame into three regions: background, head and shoulder and in the terminology of MPEG-4, there are called VOs respectively; some time it is also possible to combine the foreground part (head and shoulder) as one VO, which is also semantically meaningful.

(a) (b) (c)

Figure 2-13: Segmentation result example on one frame (frame 530) of the testing sequence. (a) is the original input frame. (b) is the segmented foreground VO. (c) is the segmented head VO.

To quantify the role of the shape-based adjustment module in the overall tracking system, a group of experimental data is used to compare the tracking performance with and without shape-based adjustment. The video data used is the mentioned 800-frame sequence. The accuracy of head region tracking is measured by comparing the tracked support map with a ground-truth support map, which is generated with a semi-automatic video segmentation tool [63] that we developed in our laboratory. The error rate is defined as

$$\text{error} = S_{\text{misclassified}} / S_{\text{ground-truth}},$$

where $S_{\text{misclassified}}$, $S_{\text{ground-truth}}$ represent the size of the misclassified and the ground-truth head region, respectively. Fig. 2-14 illustrates the result of this experiment (only the results for 100 frames, from frame 500 to 599 are included) .

In Fig. 2-14, the horizontal axis is the frame number and the vertical axis is the head region tracking error rate. The dark curve represents the error rate of the tracking algorithm without shape-based adjustments and the light curve represents that of the algorithm with shape-based adjustments. For better observation, we also overlay a bar graph on the bottom of the figure that represents the detection of canonical shapes. On this bar graph, each bar along the horizontal axis means a detected canonical shape at the frame position. It can be seen that canonical shapes are detected on about 25 percent of the 100 frames. Due to this detection result, the tracking result using shape information is better than the result of the algorithm without it. This relation is especially obvious on frames from 550 to 600, where blob tracking errors get accumulated because of the large motion of head-and-shoulder foreground and the similar color of head and shoulder regions (the dark curve). However, with canonical shape based adjustments, the tracking is much more reliable (the light curve)[5].

In Fig. 2-15, we compare the effect of boundary relaxation and 3D spatial/temporal median filter. The first row from left to right are three consecutive original frames. The second row are their segmented results without relaxation and filtering. The third row are the final results with both relaxing and filtering. We can see that in the second row, some noises on the boundary are produced because parts of the background color are very similar to that of the foreground model. However, with boundary relaxation and 3D filter, the boundaries in the third row are much smoother, both spatially and temporally.

[5]In order to give the readers better sense of the accuracy and reliability performance of our algorithm, we put frame-by-frame segmentation results of all the 800 frames on our web as well.

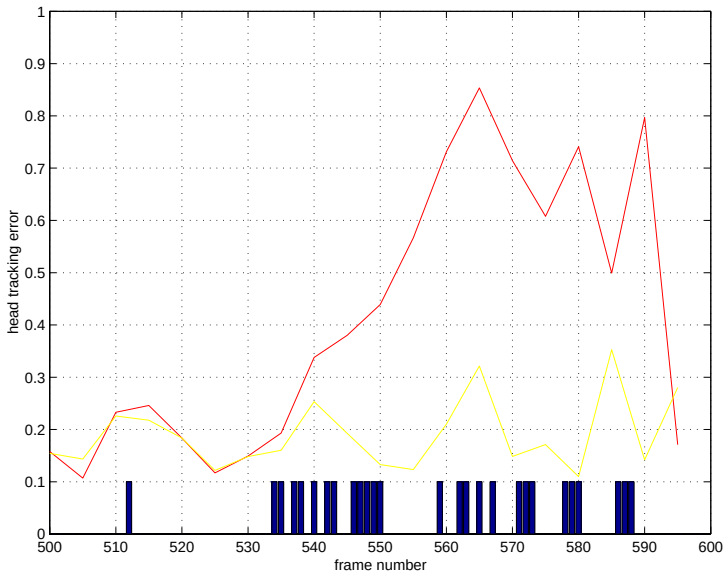


Figure 2-14: Illustration of the function of the shape-based adjustment module. The dark curve is the head region tracking error of the tracking algorithm without shape adjustment and the light curve is that of the tracking algorithm with shape adjustment. On the bottom, a bar graph is overlaid on the same temporal scale. Each bar represents that a canonical shape is detected at the frame position.

Figure 2-15: Comparison of boundary relaxation and 3D filter effects. The top row from left to right are three consecutive original frames. The second row are their segmented results without boundary relaxation and 3D filtering. The third row are the final results with both relaxing and filtering.

2.6. Application Discussion

From the experimental results, we believe the discussed segmentation algorithm is useful for introducing content-based processing for videoconference applications. One direct application is to generate video objects for MPEG-4, so that videoconference and videophone services can be built up on the MPEG-4 framework. In addition, the content information might also contribute to traditional coding framework such as H.263. The basic idea is that from the segmentation result, we have a subjective idea of the input image before we actually encode it. We can then allocate more bits to the head region macroblocks (MBs) and less to the background MBs. This way, we can use the bits more efficiently than a uniform MB encoding that a standard H.263 encoder uses. For technical reasons (MPEG-4 application is straight-forward), we discuss the H.263 application in more detail.

2.6.1 The Idea of Encoder Optimization

In the general framework of MPEG and H.261/H263 standards, there has been extensive research on optimal encoder design. This includes all the "non-standardized" steps such as coding model choice, bit allocation, adaptive quantization and rate control. The possible different choices of all these steps provide large room for optimization. In this work, we try to incorporate subjective factors, which are obtained with the previously discussed segmentation algorithm, into a traditional rate-distortion model and design an optimal H.263 compatible encoder in this sense.

Early relevant research can be found in [28, 29, 52]. In [29], an ellipse detection algorithm was used to detect the face region in typical head-and-shoulders videophone sequences and two techniques, *i.e.*, buffer rate modulation and buffer size modulation were used in a H.261 compatible encoder to control the quantization that allocated more bits to the facial region. [52] extended this idea in two aspects:

one was that it classified the image into four regions: eye, lip, face and background rather than two (face and background) as in [29] and assigned different subjective importance to each region. The other was that it introduced the idea of temporal scalability that used different temporal updating rates for different regions. For example, the lip region was updated at the highest rate in order to get lip synchronization. The ideas in these papers are interesting but we also find open problems. First, there is no uniform way to judge the rate-quality relationship and to choose proper trade-offs between rate and distortion in these rate control algorithms. Second, excessive segmentation of an image into multiple regions in these algorithms not only increases analysis complexity, but also brings visual artifacts. Applying different temporal rates on three different facial regions can in some cases be annoying when the decoded video is played back in real time.

With all these in mind, we design our region based H.263 compatible encoder, which we refer to as *region based encoder* in this thesis. As a comparison reference, we first briefly introduce Telenor's H.263 Testing Model 5 (TM5) [45] in Section 2.6.2 and then describe our algorithm in Section 2.6.3.

2.6.2 Telenor's TM5

Telenor's implementation of TM5 supports most the coding modes of H.263. Here we focus on its (online) rate control mechanism, which is implemented as follows.

1. The first intra picture is coded with $Q = 16$ (default value, can be set by user), and the buffer content is initialized as:

$$\text{buffer} = R/F_{target} + 3R/FR, \quad \text{and } B_{t-1} = \bar{B}, \tag{2.31}$$

where R is the target bit rate, FR is the frame rate of the source material (typically 25 or 30 Hz), F_{target} is the target frame rate, B_{t-1} is the bits spent

in the previous frame and $\bar{B}$ is the target number of bits for each frame (under uniform allocation).

2. For the following pictures the quantizer is updated at the beginning of each new macroblock line:

$$Q_{new} = \bar{Q}_{t-1}(1+\Delta_G+\Delta_L),\ \Delta_G = (B_{t-1}-\bar{B})/(2\bar{B}),\ \Delta_L = 12(\sum_{k=1}^{i} B_{t,k}-(i/M)\bar{B})/R, \tag{2.32}$$

where $\bar{Q}_{t-1}$ is the mean quantizer in the previous frame, M is the number of macroblocks in one frame, $B_{t,k}$ is the bits spent for the k-th MB in the current frame (time t) and i is the index of current macroblock.

3. The buffer content is updated according to the following pseudo code:

```
buffer=buffer+B_t;
while( buffer>3R/FR )
     { buffer-=R/FR; frame_incr++; }.
```

Note that the variable `B_t` is just previous B_t and the variable `frame_incr` controls the number of frames skipped from the input video.

The simple design consideration behind this scheme is that the available bits are allocated uniformly to every frame and every MB. A linear feedback is used to control the quantizer to make the actual bit consumption meet the budget (in Eq. (2.32), Δ_G controls the bit allocation to frames and Δ_L controls the allocation to MBs). If a frame uses more bits than allocated, some succeeding frames are skipped in order to maintain the buffer content. Thus this scheme may not provide a fixed frame rate due to the frames skipped, though it accepts a target frame rate as an input.

2.6.3 Region Adaptive Bit Allocation and Rate Control

In our work, we try to improve the rate control module of Telenor's implementation of H.263 while still maintain bit stream compatibility. More specifically, our contribution comes in three aspects: bit allocation, temporal adaptation and adaptive quantization.

2.6.3.1 Bit Allocation

There are two steps in bit allocation: one is to allocate bits to frames and the other is to allocate bits to macroblocks. In H.263, there are only I and P frames but no B frames[6]. Because there is no random access requirement and H.263 is designed for real time applications, P frames constitute the majority in an H.263 stream and I frames are only inserted to compensate for channel errors. It is not possible to pre-segment the input video into frame groups and assign bits to each I and P frame according to their relative complexity measures as in [102]. Therefore, in this work, we just allocate bits uniformly to each frame as TM5 does while concentrating on the controlling of bit allocation to different macroblocks.

Problem: The problem can be expressed as follows. Given the bit budget B, find the optimal bit allocation B_i to M macroblocks ($i = 1, 2, \cdots, M$) that minimizes the overall distortion in an R-D sense. As [54] pointed out, the bit budget B includes three portions: $B = B_s + B_m + B_c$, where B_s is the bits needed for header information, B_m is the bits needed to code the motion vectors, and B_c is the bits for DCT parameter coding. The portion of bits that we can control is only B_c. In [54], the bit allocation between B_c and B_m is also discussed (this means motion

[6]Or more appropriately, *intra* and *inter* MBs, but in this work we only consider intra and inter transitions at the frame level. In addition, only baseline mode, no advanced modes like the PB mode are considered in our work.

compensation should also be controlled). We do not consider this problem here. In the remaining part of this chapter, B and B_i refer to DCT bits only.

Distortion Model: In H.263, the DCT coefficients within a macroblock are quantized with one quantizer Q_i. Assuming the DCT coefficients distributed uniformly within one macroblock, we use the following expression to measure the coding distortion for one frame:

$$D = \sum_{i=1}^{M} \beta_i (k{Q_i}^2), \tag{2.33}$$

where $k{Q_i}^2$ refers to the objective quantization errors with k being a constant and Q_i being the quantizer, β_i is a subjective importance factor for macroblock i. In this work, we decide β_i for each MB according to the support map $s(x, y)$ obtained through our segmentation algorithm, *i.e.*, we assign higher β to those MBs labeled as head region and lower β to those labeled as shoulder and background regions.

Encoder Model: In this work, we use the following equation to model the functional relationship between an H.263 encoder's bit consumption B_i and its quantizer Q_i:

$$B_i = a_i \sigma_i^2 Q_i^{b_i}, \tag{2.34}$$

where σ_i^2 is the (motion compensated) standard deviation of the luminance of the macroblock, a_i and b_i are two constants. According to our experiment as well as results reported in available papers [102, 26, 108], the empirical value of b_i is in the range of -1.5 to -2. In practice, we find $b_i = -2$ is a good compromise between computational complexity and modeling accuracy. In addition, because most motion estimation modules produce SAD (standard absolute deviation) as a byproduct, we use MAD (mean absolute deviation) in place of the standard deviation σ to save

computation time. Here the definition of SAD and MAD are

$$\text{SAD} = \min_{dx,dy}\Big\{\sum_{x=1}^{x=16}\sum_{y=1}^{y=16}|\text{original}(x,y)-\text{previous}(x+dx,y+dy)|\Big\}, \quad (2.35)$$

$$\text{MAD} = \frac{1}{256}\text{SAD}, \quad (2.36)$$

where (dx, dy) is the motion vector, and original(x, y) and previous(x, y) refer to pixel at the position (x, y) in the current and the previous frames, respectively. More specifically, we use

$$B_i = a_I \text{MAD}_i^2 Q_i^{b_i} \quad (2.37)$$

to approximate Eq. (2.34) in our implementation. In our experiments, we find that the behavior of Eq. (2.34) and that of (2.37) are quite similar but the latter one is much easier to obtain. In addition, because both of them are empirical rather than theoretical, the accuracy of modeling also depends on proper choice and adaptation of model parameters like a_i, which we will discuss later.

R-D Optimization: With distortion model Eq. (2.33) and encoder model Eq. (2.34), it is easy to solve the R-D optimization problem with the Lagrange method:

$$S = D + \lambda B = \sum_{i=1}^{M} D_i + \lambda \sum_{i=1}^{M} B_i. \quad (2.38)$$

Setting $\frac{\partial S}{\partial B_i} = 0$, we have:

$$\frac{\beta_i a_i \sigma_i^2}{B_i^2} = -\frac{\lambda}{k} = constant. \quad (2.39)$$

If we let

$$w_i = \sqrt{\beta_i a_i \sigma_i^2}, \quad (2.40)$$

then the optimal bits allocation to each MB can be obtained as

$$\hat{B}_i = \frac{w_i}{\sum_{k=1}^{M} w_k} \bar{B}, \tag{2.41}$$

where $\bar{B}$ is the average bits allocated to each frame. This equation shows that the optimal bit allocation $\hat{B}_i$ to MB i is proportional to its subjective important factor β_i and its objective complexity σ_i^2, which is in accord with our intuition.

2.6.3.2 Temporal Adaptation

In the bit allocation weight equation (2.40), the subjective factor β_i is used to control the bit allocation so as to introduce spatial adaptation. In most videoconference cases, the background is relatively static and it is reasonable to reduce its temporal updating rate while still maintaining good visual quality. The benefit is that we can avoid spending bits on visually unimportant information. In some cases, for example, when the background itself is static, the bits we used for the background account for nothing but white noise. Even when there are changes in the background, it does not bring much visual degradation to remove some temporal high frequencies in the background.

Under this consideration, we classify the P frames in H.263 into PO and PF frames. In PO frames only the foreground object MBs are coded, while in PF frames all the MBs in the full frame are coded. PF frames work as anchors for PO frames and their temporal relation is illustrated in Fig. 2-16 where possible I frames are also included. The parameter T_{PF} (time between two PF frames) should be adjusted according to the estimation of background motion. In this work we measure T_{PF} as the number of PO frames between two adjacent PF frames. Also notice that PO and PF control can be easily integrated into bit allocation Eq. (2.41) by assigning $\beta_{background} = 0$ for PO MBs and $\beta_{background} \neq 0$ for PF MBs.

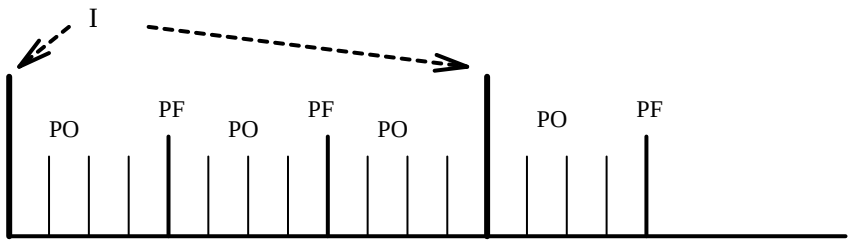


Figure 2-16: Temporal scalability: PO and PF frames illustration. At the PO frames, only the foreground object's MBs are coded, while at the PF frames, all the MBs of the whole frame are coded. Thus the temporal updating frequency of the background is lower than that of the foreground.

2.6.3.3 Adaptive Quantization

In the above discussion, we derive an optimal bit allocation to each MBs but how to choose proper quantizers to achieve the allocation remains a problem. Actually, since the encoder model is only an empirical one, there are no clear theoretical functions like $Q = f(B)$ or $B = f(Q)$. In addition, due to the real time nature of H.263, it is not appropriate to introduce trial coding to estimate quantization model parameters as was suggested in [26]. There should always be some online feedback mechanism in the rate control module in order to make sure that the actual bit rate does not exceed the available bandwidth. In this work, we propose two adaptive quantization schemes to realize our bit allocation.

Scheme One: Scheme one is developed based on Telenor's TMN5 model. We adapt Telenor's feedback rate control scheme as follows:

$$Q_{t,i} = \bar{Q}_{t-1,s(i)}(1 + \Delta_G + \Delta_L), \tag{2.42}$$

where

$$\Delta_G = (B_{t-1} - \bar{B})/(2\bar{B}), \tag{2.43}$$

$$\Delta_L = 4(\sum_{k=0}^{i} B_{t,k} - \sum_{k=0}^{i} \hat{B}_{t,k})/R. \quad (2.44)$$

Compared with Telenor's scheme Eq. (2.32), we maintain different average quantizers $\bar{Q}_{t-1,s(i)}$ for different segmented regions $s(i)$. The idea is that the quantizers should be similar between neighboring frames for the same type of regions. Another change is that we use a different cumulative target rate $\sum_{k=0}^{i} \hat{B}_{t,k}$ to generate the local adaptive factor, not a uniform allocation any more.

In the encoder model Eq. (2.34), parameter a_i is assumed to be MB dependent. In the implementation of *scheme one*, we try to classify the MBs into different groups according to their complexity σ_i^2:

$$g(i) = \frac{\sigma_i^2}{G_{th}}, \quad (2.45)$$

where G_{th} is a threshold to divide MBs into groups and $g(i)$ is the group index of macroblock i. The average a_i is tracked and updated for each group individually as $\bar{a}_{t,g(i)}$. When encoding MB i at the time t, we have

$$a_{t,g(i)} = \bar{a}_{t-1,g(i)}. \quad (2.46)$$

Note that in quantization model Eq. (2.34) we set b_i as -2 and only a_i is used to account for input adaptation. By grouping the MBs by their complexity and estimating model parameters individually for each MB group, we can better control the performance of this empirical model.

Scheme Two: Another way to control the quantizer Q_i is to make use of the empirical encoder model Eq. (2.34) directly. From Eq. (2.34), it is easy to derive

the Q_i expression in relation to bit allocation B_i:

$$Q_i = \sqrt{\frac{a_i \sigma_i^2}{B_i}}. \tag{2.47}$$

So according to this empirical encoder model, we can allocate optimal B_is as well as the Q_is that are used to realize the optimal bit allocation. In practice, in order to fit the empirical model to actual input data distribution, we have to add a linear feedback term and set the actual quantizer as:

$$\tilde{Q}_i = \sqrt{\frac{a_i \sigma_i^2}{B_i}}(1 + \Delta_G + \Delta_L), \tag{2.48}$$

where the linear term $(1 + \Delta_G + \Delta_L)$ is as defined in Eq. (2.32). In addition, the encoding model parameter a_i is also adapted by groups as in scheme one.

Implementation Issues: In H.263, the encoding quantizer can be adjusted at three layers: picture layer (QUANT), group of block (GOB) layer (GQUANT) and macroblock (MB) layer (DQUANT). Among them, both QUANT and GQUANT are 5-bit absolute values, while DQUANT is a 2-bit value that reflects the quantizer difference from the previous one. It is not, therefore, possible to set the quantizer of a macroblock arbitrarily as indicated in the previous two adaptation algorithms. The practical implementation is realized in a constrained domain if a bitstream compatibility is desired.

In our implementation, macroblocks are processed by GOBs. Each GOB contains one horizontal stripe of MBs. First for each MB i within the current GOB, its Q_i is calculated according to equation (2.42) or (2.47). An average quantizer is then obtained by

$$\bar{Q} = \sum_{i \in \text{GOB}} Q_i / \sum_{i \in \text{GOB}} 1. \tag{2.49}$$

The GOB quantizer is set to $\bar{Q}$ and DQUANT for each MB is then set in a best effort way.

In videoconference applications, the MBs in a GOB (a horizontal MB stripe) tend to belong to two classes: either the background and the head regions or the background and the shoulder regions. Because the background is assumed to be relative static, background MBs are skipped in most of the cases and they do not influence the GOB quantizer $\bar{Q}$ as defined in equation (2.49). Therefore, $\bar{Q}$ is obtained only for one class of MBs, either head region MBs or shoulder region MBs. These MBs tend to have similar quantizers and the 2-bit DQUANT is enough to represent their difference.

2.6.4 Experimental Results

We implemented a simulation version of our region-based H.263 encoder on a PC with a hardware system as described in Section 2.5.. As mentioned already, because of the online feature of our segmentation algorithm, we could not use standard video sequences in our testing. Instead we used the 800-frame testing sequence we described in Section 2.5. and all the experiments in this chapter were based on this testing sequence.

In the experiment, our region based algorithm was compared with Telenor's TM5 [100]. In our algorithm, two different adaptive quantization schemes are used to realize the optimal bit allocation budget as was discussed in Section 2.6.3.3. In the following discussion, we refer to these two algorithms as *scheme one* and *scheme two*. We use *tmn-2.0* to represent Telenor's standard H.263 implementation because their latest software implementation is Version 2.0. In addition, all the three algorithms used in the experiments used the baseline mode, no advanced modes are used. The target frame rate was 10 fps, and the bit rate was set to 32 kbps (kilo bits per

second). The parameters used in our algorithm were $G_{threshold} = 200$, $T_{PF} = 30$. The subjective factors used were $\beta_{head} = 4$, $\beta_{shoulder} = 1$, $\beta_{background} = 0$ for PO frames and $\beta_{head} = 1$, $\beta_{shoulder} = 1$, $\beta_{background} = 1$ for PF frames.

Fig. 2-17 shows the bit allocation of a typical PO frame by one of our region based algorithms (*scheme one*). We number the images from left to right and top to bottom. Fig. 2-17(a) is an original frame (the 295th frame of the testing sequence), Fig. 2-17(b) is the grayscale display of the MAD (mean absolute deviation) of MBs (the brighter, the bigger), Fig. 2-17(c) is the bit allocation budget according to Eq. (2.41) and Fig. 2-17(d) is the actually achieved bit allocation by the adaptive quantization scheme one, which was discussed in Section 2.6.3.3. Note that (d) is not strictly equal to (c) due to the error in the empirical encoder model and the nature of the feedback control mechanism that is used to fit the model with the actual data.

To better evaluate the encoder quality, we introduce the concept of *region-based PSNR* and define it as

$$\text{PSNR} = 10 \log_{10} \Big\{ \frac{255^2 \mathcal{S}(s_k)}{\sum_{s_k(x,y)=1} [I(x,y) - \hat{I}(x,y)]^2} \Big\},$$

where $I(x,y)$ and $\hat{I}(x,y)$ are the original and the decoded image pixel's scalar value (only luminance factor Y is considered in this experiment), $\mathcal{S}(s_k)$ is the size of blob k's support map and can be defined as

$$\mathcal{S}(s_k) = \sum_{(x,y) \in \mathcal{I}} s_k(x,y).$$

With region-based PSNR, we try to compare the decoded image quality of our algorithm with that of the Telenor's standard H.263 algorithm. The testing sequence we used for explaining here is a 100-frame sequence (from the 220th to the 320th

frame of the 800 frame testing sequence). To have a fair comparison of the rate control mechanism, we treat the decoded video frame by frame. That is, for those frames skipped by the encoder, their PSNR is calculated with the repeated previous frame at the decoder. Fig. 2-18 shows the frame by frame comparison of the head region PSNR of the three algorithms and Fig. 2-19 is the frame by frame comparison of the regular PSNR (calculated over the whole frame) of three algorithms. From these two figures, we see that our algorithms (both *scheme one* and *scheme two*) improve the head region PSNR by about 1 to 1.5 dB in the whole testing frame range as compared with *tmn-2.0*. This gain comes at the expense of the loss of the regular PSNR of our algorithms by about 0.5 to 1 dB, as compared with that of *tmn-2.0*. The regular PSNR loss is due mainly to the PSNR loss in the background region, which is always the biggest region in the image and thus has the largest influence on the overall PSNR result. However, we also notice that within the comparing frame range, *tmn-2.0* skips more frames than our algorithms, which is reflected in the frequent negative peaks in its PSNR curve. In Fig. 2-20 the bit rate produced by the three algorithms are compared. The curves indicate that both *scheme one* and *scheme two* produce smoother rate than *tmn-2.0*. That means the alternation of PO and PF frames scheme in our region-based algorithm does not bring higher bit rate fluctuation than *tmn-2.0*, and the feedback rate control scheme we used works well. In Fig. 2-20, the average bit rate of *tmn-2.0* is higher than that of our two algorithms. That is because it skips more frames and thus has more bits to use for each coded frame. In addition, when we compare our *scheme one* and *scheme two*, we can see that *scheme one* is a little bit better than *scheme two* on the smoothness of the bit rate produced, nevertheless their PSNR behaviors are quite similar.

To evaluate the PSNR performance more accurately, we list the frame by frame average of the PSNR within the 100-frame range for all the three algorithms in

	coded frames	rate (kbits/s)	PSNR	PSNR-head	PSNR-s	PSNR-b
Scheme one	91	31.84	32.86	33.66	29.72	35.75
Scheme two	90	32.04	32.84	34.02	29.25	36.40
Tmn-2.0	79	32.26	32.98	32.75	29.94	36.51

Table 2.1: Comparison of compression efficiency of our algorithm and that of the standard H.263. Rate is in kilo bit per second, PSNR is the peak SNR of the entire frame in dB, PSNR-b is PSNR of the background region, PSNR-h is PSNR of the head region and PSNR-s is the PSNR of the shoulder region.

Table 2.1. In Table 2.1, we can see that both *scheme one* and *scheme two* encode about ten more frames than *tmn-2.0*. This means as a rate control algorithm, both *scheme one* and *scheme two* are better than *tmn-2.0*. In addition, due to their region adaptive bit allocation, both our algorithms produce higher head region PSNR than *tmn-2.0*, the difference is about 1 dB, averaged over the 100 frames. In the regular PSNR sense, our algorithms' performance is only about 0.2 dB below that of *tmn-2.0*. Unlike the regular PSNR curve in Fig. 2-19, the average regular PSNR drop of our algorithms is less because *tmn-2.0* skipped more frames, which penalized itself in the average performance evaluation. Obviously, because of the relative amount of PSNR changes, we believe this result of our algorithm is surely interesting for most videoconference applications.

Fig. 2-21 compares two reconstructed frames by the region based algorithm *scheme one* (left) and *tmn-2.0* (right). The region-based algorithm exhibits better subjective quality in the head and facial region. In addition, the PO and PF alternation maintains decent background quality and the overall trade off is beneficial to the subjective quality evaluation.

Because we did not have source code for a real time H.263 encoder, we did not implement a real time system that combines the segmentation and H.263 encoding. In our experiment, we used Telenor's H.263 source code, which itself is for simulation purpose and needs much optimization for real time performance. Thus an

experimental result for the complexity of our region based H.263 encoder is not yet available. However, as our segmentation algorithm runs at 30 fps for sub-sampled QCIF size video and most commercial H.263 software run at around 20 fps for QCIF video on average Pentium PCs, it is reasonable to believe that we can combine the segmentation and H.263 encoding into one system and still attain the frame rates around 8-10 fps for QCIF video. This is work we expect to do in the near future.

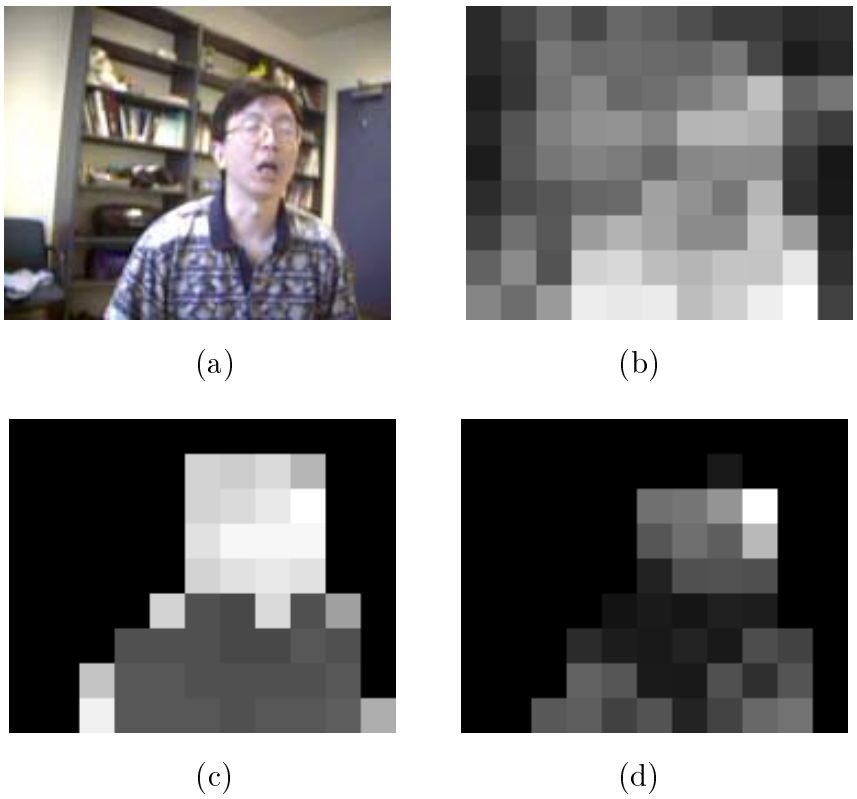


Figure 2-17: Bit allocation of a typical PO frame by one of our region based algorithm (*scheme one*). (a) is an original frame, (b) is the grayscale display of the MAD (mean absolute deviation) of MBs (the brighter, the bigger), (c) is the bit allocation budget according to equation (2.41) and (d) is the actual achieved bit allocation by the adaptive quantization scheme one.

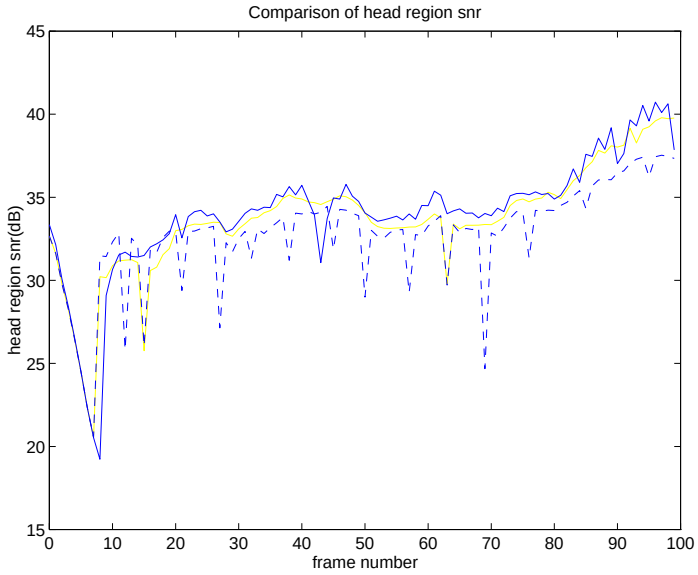


Figure 2-18: Frame by frame comparison of the head region PSNR of three algorithms. The light solid curve is for *scheme one* algorithm; the dark solid curve is for *scheme two* algorithm and the dashed curve is for *tmn-2.0* algorithm.

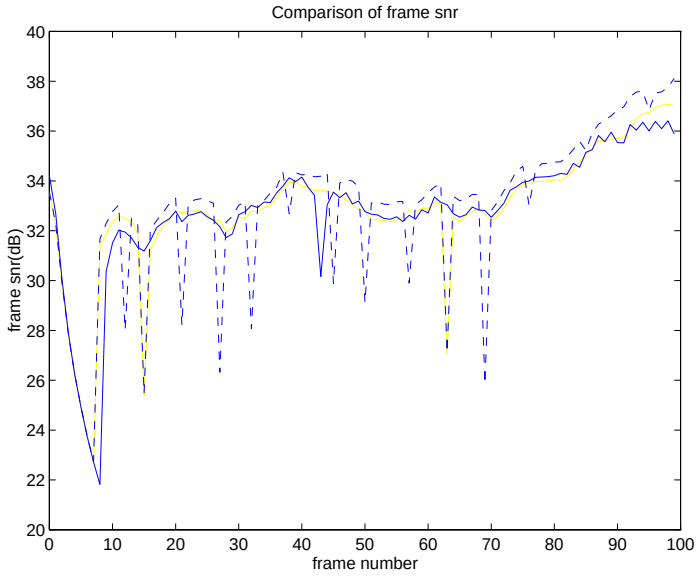


Figure 2-19: Frame by frame comparison of the frame level PSNR of three algorithms. The light solid curve is for *scheme one* algorithm; the dark solid curve is for *scheme two* algorithm and the dashed curve is for *tmn-2.0* algorithm.

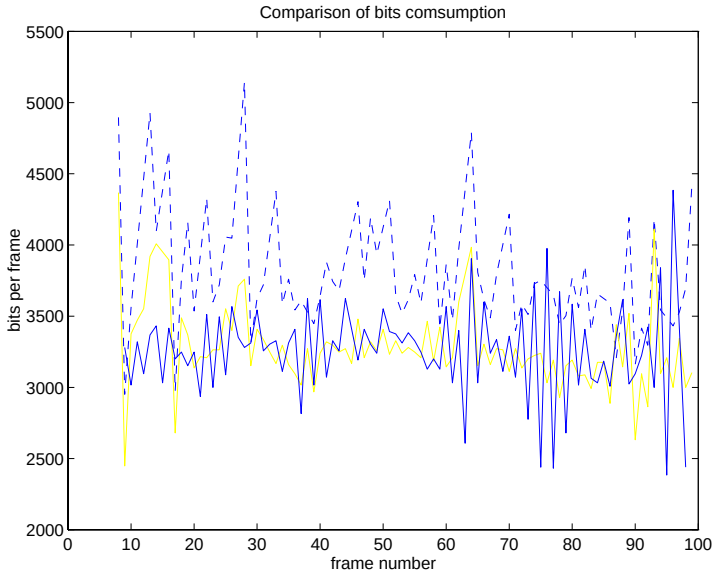


Figure 2-20: Comparison of the bit consumption of the three algorithms. The light solid curve is for *scheme one* algorithm; the dark solid curve is for *scheme two* algorithm and the dashed curve is for *tmn-2.0* algorithm. The average bit rate of *tmn-2.0* is higher than that of our two algorithms. That is because it skips more frames and thus has more bits to use for each coded frame.

2.7. Concluding Remarks

In this chapter, we proposed a simple online video segmentation algorithm for videophone applications. This algorithm uses feasible limitation on its application domain but gets good results with relative low computational complexity. Because of its low complexity and real time performance, this segmentation algorithm is especially useful for introducing content-based processing in real time video coding. One direct application of this segmentation algorithm is MPEG-4 VO creation. Another application is to apply this segmentation algorithm to traditional DCT based video coders. In our experiment, an H.263 compatible simulation encoder is implemented that makes use of the segmentation results in its rate control. Spatial and temporal adaptation are introduced and an improvement in subjective quality is observed. Our work shows that it is possible to combine the segmentation and traditional

coding systems into an integrated intelligent coding system while still maintaining real time performance on an average PC platform.

(a) (b)

Figure 2-21: Comparison of a typical pair reconstructed frames. (a) is the reconstructed 260th frame (of the testing sequence) with algorithm *scheme one*; (b) is the reconstructed 260th frame with *tmn-2.0*. The facial region of the left image (a) is clearer than that of the right image (b).

An interesting point worth noticing is that we can use the discussed segmentation algorithm as *an optional module* for traditional videoconference systems. That is, when the segmentation module loses track or can not fit the detected foreground to its internal blob model (for example, when there are multiple persons in the scene), it turns itself automatically back to the initialization loop and the H.263 encoder in the system works in its regular mode. However, when the segmentation module finds its expected foreground successfully and changes back to the tracking loop, the segmentation output is then available as an option to support region based rate control for the H.263 encoder. This way, our segmentation algorithm can be incorporated into general-purpose videoconferencing systems without any concern that the tracking failure of the segmentation module will bring any negative influence to the whole system.

Chapter 3

Automatic Model-based Anchorperson Detection

3.1. Introduction

The amount of multimedia data is increasing every day with a rapid speed around the world. The growing requirement for efficient indexing and management of multimedia data becomes an urgent problem. Though a fully automatic indexing and management approach is still hard to achieve in the general case, many research projects have already obtained good results on constraint domain problems, for example, automatic analysis of broadcast news videos. So far, the most optimistic direction for automatic news indexing and management seems to be combined multimodal signal analysis, *i.e.*, the combined analysis of video, audio, as well as close caption data [34, 61]. In this chapter, we discuss our work on visual information analysis, especially anchorperson detection[1].

The role of anchorperson detection for broadcast news management has already been discussed by several researchers. For example, Zhang [32] developed a news indexing system based on anchorperson detection. In his work, simple temporal

[1]Part of this work was finished when the author stayed as a summer intern with AT&T Labs. It is, therefore, part of a comprehensive news analysis project [34], which tackles video, audio and text in order to obtain optimal analysis results.

and spatial templates are designed and matched to actual video frames. Detected anchorperson frames are used as temporal transition positions for indexing of individual news story units. In [31], Hanjalic *et al.* also discussed an anchorperson detection algorithm based on key frames obtained from scene change detection algorithms. Their major observation is that anchorperson frames are always similar in their spatial setup and they also have the largest temporal frequency among the key frames. The common limit in these approaches, however, is that they all have strong assumptions on patterns to be detected. In Zhang's approach [32], anchorpersons can only appear in certain positions and the setup of the frame background is also fixed. While in [31], the backgrounds cannot change for the frames to be detected as anchorperson frames.

In this chapter, we design a new anchorperson detection algorithm that works in much more general cases, *i.e.*, we do not assume the fixed positions of anchorpersons or static backgrounds. In our work, the anchorpersons are models with respect to two aspects: a spatial model and a temporal model. The spatial model is created to represent anchorperson on individual frames. In order to accommodate variations between different anchorperson frames as well as capture consistency among them, a statistical model is designed with respect to the shape and color distribution of anchorpersons. The anchorperson detection problem is then modeled as a statistical pattern detection problem. Intuitively, the anchorperson detection problem is related to the real-time videophone segmentation work discussed in Chapter 2 in that the objects can be represented with the same head-and-shoulder pattern. Therefore, the same shape model can be used. However, the major difference here is that the videophone segmentation problem is an online one while the anchorperson detection problem is an offline one. To fit the model with a video frame in an offline matter, extra efforts are needed to reduce the complexity of the high dimensional searching

problem to a manageable level.

In addition to its spatial model design, our algorithm detects anchorpersons by imposing a temporal model analysis module right after the spatial model analysis module. This is illustrated in Fig 3-1. Their relation is that the spatial module detects frames containing "head-and-shoulder" patterns, while the temporal module further analyzes the temporal frequencies of these "head-and-shoulder" patterns. Only those patterns with high temporal frequencies are detected as anchorpersons. As indicated in Fig 3-1, the overall processing modules of the system are cascaded so that the handling of spatial and temporal information is separated, which is more efficient than previous approaches [32, 31]. As is shown in Fig 3-1, the spatial module, *i.e.*, the head-and-shoulder model fitting is carried out on individual frames. This is more efficient as compared with a model fitting in both spatial and temporal domains. Systematically, this model-fitting module is placed right after a scene-change detection and key frame detection module. The input video is first filtered with the key-frame detection module, and then the anchorperson detection module runs only on the detected key frames. Because the key frames constitute only a fraction of the overall video data, the anchorperson detection algorithm can be designed on top of fully decoded image frames without incurring much computation and storage burdens. As for the key-frame detection module, algorithms that run in the MPEG compressed domain such as [74, 75] can still be used to gain computational efficiency. In addition, the temporal analysis module is applied only to the positive output frames of spatial analysis module, which makes the whole detection algorithm very efficient.

We develop this chapter by discussing the spatial model design and the solution of the complexity problem in model fitting in Section 3.2.. In Section 3.3. we describe temporal model analysis module. In Section 3.4. we present experimental results on

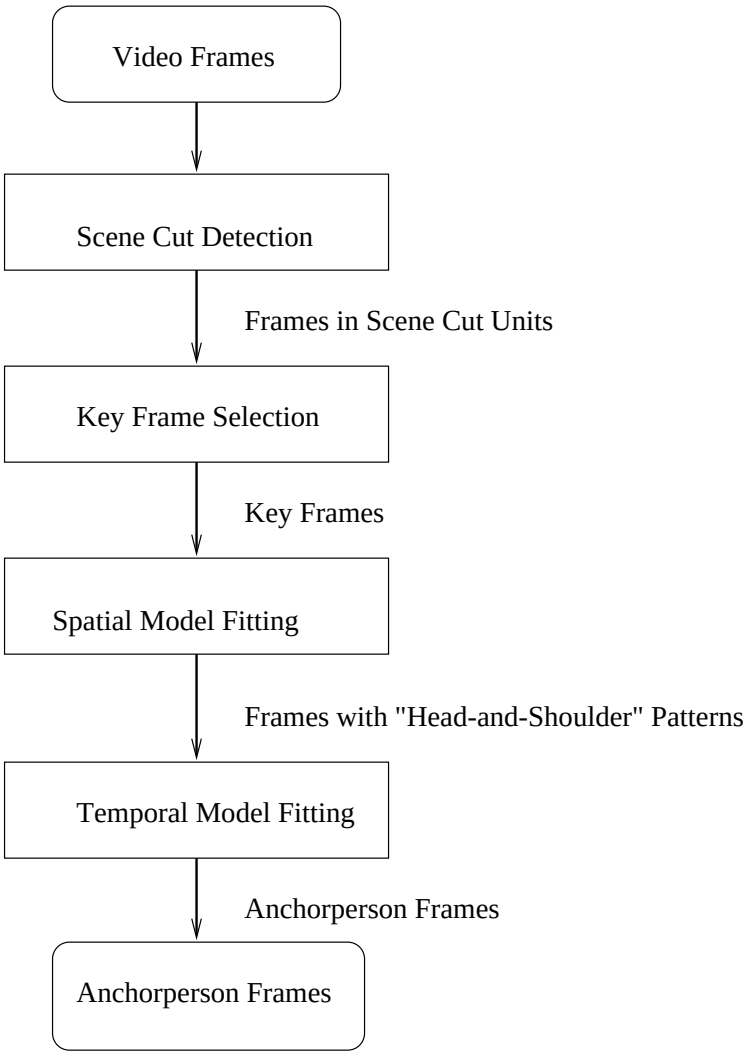


Figure 3-1: Illustration of the relationship of the spatial analysis module, temporal analysis module and the related preprocessing modules

actually video data from NBC Nightly news, and finally in Section 3.5. we conclude the chapter.

3.2. Spatial Module: Statistical Model-based Pattern Detection

3.2.1 The Model Design

Based on the real time segmentation work as discussed in Chapter 2, a straightforward solution for anchorperson detection problem is to apply the blob model and the shape model we developed there directly to the problem. However, for news video analysis problem, we have to work offline with no clues to build up a background model. In addition, the blob model does not help much neither for *detection problem* as it did for *tracking problem*. Only the shape model remains valid. But to apply the shape model as developed in Section 2.3.2 to the detection problem, the shape model has to be searched over the video frame in different scales and at different positions [20], which is not a feasible approach.

In order to reduce searching complexity, we combine shape detection with a face detection problem, *i.e.*, in addition to fitting to a head-and-shoulder type shape model, we require that the anchorperson object to be detected also have a face region that returns positive by our face detector. This way, the scale and the position in which the shape model to be used for shape fitting can be normalized with respect to the detected face regions, *i.e.*, their sizes and positions. This idea can be better illustrated by Fig. 3-2, in which the rectangle represents the face region. The contour of the object is first normalized with respect to the scale and position of the face region and then compared to the head-and-shoulder shape model.

As such, the anchorperson detection problem is then decomposed into two sub-

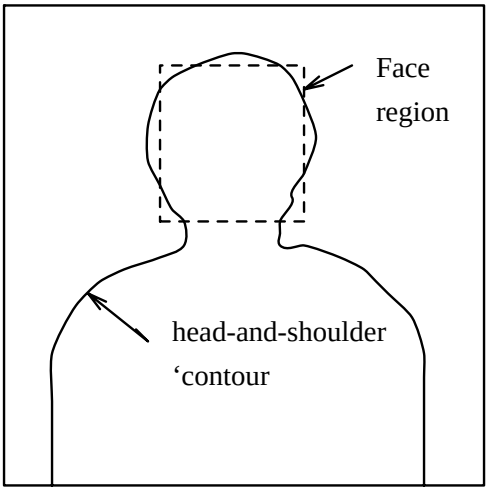


Figure 3-2: Illustration of the modeling of anchorpersons with a face model and a head-and-shoulder shape model.

problems: a face region detection and a statistical shape model fitting.

3.2.2 Face Region Detection Problem

Face region detection is a problem discussed by many papers in the literature. A simple but reliable approach as reported by many papers [105, 39, 5, 112, 2] is skin color detection. In this work, we take a skin color detection approach similar to [105]. Color is modeled within the projected (Cb, Cr) space from the (Y, Cb, Cr) space. The skin color vector $\mathbf{v_c} = (C_b, C_r)$ is assumed to have a 2D Gaussian distribution:

$$P(\mathbf{v}) = \frac{1}{\sqrt{2\pi}|\Sigma_c|^{1/2}} \exp\{-(\mathbf{v}-\mathbf{C}_0)\Sigma_c{}^{-1}(\mathbf{v}-\mathbf{C}_0)^T\}$$

where the average vector $\mathbf{C}_0$ and covariance matrix Σ_c are obtained via training examples. Unlike the Bayesian approach used in [105], we take a simple threshold on $P(\mathbf{v})$ for each pixel and a binary image is produced to represent the location of skin color regions. After that, morphological filters are employed to remove the noise

regions as well as to merge adjacent regions. The produced possible skin regions are then further filtered with a shape analyzer that imposes shape constraints, *i.e.*, a valid face region's containing rectangle should only be in a reasonable aspect ratio range. For this work, we set this acceptable range to be [0.9, 1.8).

The Gaussian skin color model is trained on three days of NBC Nightly News video data (about 200 out of 1500 frames) and then tested on another three days of data. Fig. 3-3 shows three representative output frames we have in our experimental results. Fig. 3-3(a) is a detected face region corresponding to an actual anchorperson frame, while in 3-3(b) the detected region is a face, but not a face of an anchorperson. In 3-3(c), the detected region has skin-like color, but it is not a face at all.

(a) (b) (c)

Figure 3-3: Representative results of face region detection.

To filter out the outputs in the cases of (b), (c), a shape model is further designed to find the coherency of anchorperson objects. That is, in addition to containing a face region in skin color, we assume that most anchorperson frames can be represented as a typical head-and-shoulder pattern. As in most cases, anchorpersons always face directly towards the camera, this assumption is valid in most of the time, if not all of the time, according to our observation. We impose this constraint by a Gaussian shape modeling.

3.2.3 Shape Modeling of Head-and-Shoulder Patterns

Similar to color modeling, the shape modeling in this work is also based on a Gaussian assumption and a parameter training based on valid examples. This is quite similar to the shape modeling work we used in Section 2.3.2. We formulate this modeling algorithm in three steps: vectorization, statistical representation and training.

Vectorization The vectorization algorithm is copied from Section 2.3.2 except for a face region based normalization process. As discussed in the previous section, our shape modeling is designed in addition to a skin color modeling module. Therefore, for each valid head-and-shoulder pattern, a detected containing rectangle of the face region is already available. To make the shape vectors comparable, the actual shapes are first normalized with respect to their face rectangles before any quantization is done. Because the process of normalizing removes the shape difference introduced by different scales and spatial positions, the normalized shapes can be compared directly with respect to their absolute contour positions. This comparability can be easily maintained by a simple vectorization scheme.

Given a valid anchorperson frame to be used as a positive sample of our shape modeling, suppose its detected face region is denoted by (x_0, y_0, w, h), where (x_0, y_0) is the center of the rectangle and w and h are its width and height respectively. Then the process of normalization can be represented as a linear mapping function $m(x_0, y_0, w, h)$, which transforms the detected face region into an one by one rectangle centered at (0,0).

After the shape normalizing transformation, the shape is vectorized on the transformed coordination with a stripe-based algorithm as discussed in Section 2.3.2. That is, with N stripes, the shape contour is vectorized into a $2N$ dimensional feature vector $\mathbf{v}$. If we denote the contour of the anchorperson object as $\mathbf{C}$, then the

process of vectorization can be expressed as:

$$\mathbf{v} = f(x_0, y_0, w, h, \mathbf{C}). \tag{3.1}$$

That is, the vector $\mathbf{v}$ is obtained by first normalizing C with respect to face region (x_0, y_0, w, h) and then quantizing with the stripe-based algorithm.

Statistical Representation With above vectorization algorithm, the shape vectors of the valid anchorperson patterns are modeled as a $2N$ dimensional Gaussian distribution, characterized by a mean vector $\bar{\mathbf{v}}_s$ and a covariance matrix Σ_s. In practice, the mean $\bar{\mathbf{v}}_s$ and covariance Σ_s are obtained through a set of training shapes. A sufficient measure of the likelihood is Mahalanobis distance:

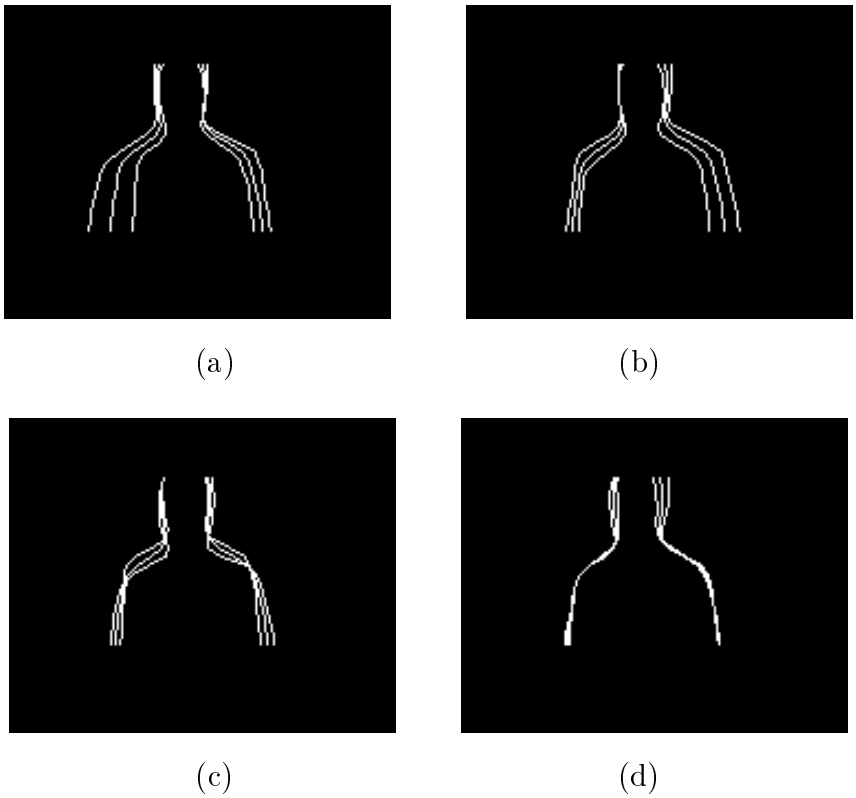
$$D(\tilde{\mathbf{v}}) = \tilde{\mathbf{v}}^T {\Sigma_s}^{-1} \tilde{\mathbf{v}}. \tag{3.2}$$

where $\tilde{\mathbf{v}} = \mathbf{v}_s - \bar{\mathbf{v}}_s$.

Training In practice, we obtain the shape vector statistics by training. Training data is collected from one week's NBC Nightly News data. Anchorpersons are segmented manually with an interactive segmentation tool [63], with totally about 140 of them segmented. Their shapes are used as positive samples.

In Fig. 3-4, we visualize the produced shape statistics, *i.e.*, mean $\bar{\mathbf{v}}_s$ and covariance matrix Σ_s by calculating the eigen-decomposition of Σ_s. In each subfigure, three shapes are overlaid that correspond to three vectors: $\bar{\mathbf{v}}_s$, $\bar{\mathbf{v}}_s + a_i\phi_I$, $\bar{\mathbf{v}}_s - a_i\phi_i$, where a_i is a weighting factor and ϕ_i is the i-th eigenvector. From subfigure (a) to (d), the eigenvectors used are the first, second, third and fourth of the system, respectively. Readers can observe the distribution of shape changing along the first several principal eigenvectors and have a general sense of the shape model built up

in this work.



(a) (b)

(c) (d)

Figure 3-4: Visualization of the shape statistics created by shape training.

3.2.4 Model Matching

So far, we have built up Gaussian models for the anchorperson pattern in the sense of its face color and shape statistics. Model matching is then a two-step process. In the first step, possible face regions are detected, which can be represented with their containing rectangles $(x_0, y_0, w, h)_i$. In the second step, for each face region a corresponding head-and-shoulder shape vector $\tilde{\mathbf{v}}$ is searched in a $2N$ dimensional feature space ($2N$ is the length of the feature vector) to maximize certain criteria,

which is easily converted to a model-based shape detection problem as follows:

$$\tilde{\mathbf{v}} = \arg\max_{\tilde{\mathbf{v}}}\{\mu_i E_{\text{internal}}(\tilde{\mathbf{v}}) + \mu_e E_{\text{external}}(\tilde{\mathbf{v}})\} \tag{3.3}$$

where E_{internal} is the model energy term that represents the shape $\tilde{\mathbf{v}}$'s similarity to our trained anchorperson shape model. We set it as $E_{\text{internal}} = D(\tilde{\mathbf{v}})$, where D is defined in Eq. 3.2. The external energy term, E_{extermal}, consists of image feature related measures that represents our subjective judgements on anchorperson image features. It is related to the shape vector $\tilde{\mathbf{v}}$ through a mapping function:

$$\mathbf{C} = f'(x_0, y_0, w, h, \tilde{\mathbf{v}}), \tag{3.4}$$

where $\mathbf{C}$ denotes the object contour and the function $f'(x_0, y_0, w, h, \cdot)$ is the inverse of the mapping function $f(x_0, y_0, w, h, \cdot)$ as specified in Eq. 3.1. Based on this mapping function, the external energy function is further defined as a function of contour $\mathbf{C}$:

$$E_{\text{external}}(\tilde{\mathbf{v}}) = E_{\text{external}}(\mathbf{C}) = \mu_1 E_{\text{gradient}}(\mathbf{C}) + \mu_2 E_{\text{symmetry}}(\mathbf{C}) + \mu_3 E_{\text{homogenity}}(\mathbf{C}).$$

In above equation, E_{gradient} is the image gradient measure on the shape contour, E_{symmetry} is the color symmetry measure and $E_{\text{homogeneity}}$ is the color homogeneity measure of the anchorperson regions defined by its contour information. Among them, the first term is the typical term for edge detection problems and the second and third terms are domain specific terms we added to improve the stability of the solution. The intuition is that the color of anchorperson regions should be symmetric as well as homogeneous. The weighting factors μ_i are determined in an empirical way.

In practice, the searching problem as expressed in Eq. 3.3 is carried out in a vector space of $2N$ dimension, where N is the number of stripes we use to subsample the anchorperson shape. To obtain good shape representation quality, N is generally chosen to be greater than 25 in our experiments. This is a typical high dimensional minimization problem, which has been discussed by a number of papers on statistical shape modeling. Cootes *et al.* [20] proposed to use an adjusting vector with each of its components corresponds to an adjustment along a normal to the model boundary toward the strongest image edge. Kass *et al.* [50] used a potential image to guide the searching process. Kervrann and Heitz [51] used a simulated annealing algorithm to locate the minimization status of their shape model. Jain *et al.* [47, 46] designed a deterministic gradient descent algorithm in their work of deformable template. In this work, our shape model as defined in Eq. 3.3 includes grayscale gradient information as well as additional constraints such as symmetry and homogeneity, which are hard to be represented as a gradient vector or potential fields. Therefore, we use a more general sense minimization algorithm: *downhill simplex algorithm* [30]. An interesting feature of this algorithm is that it is purely based on the evaluation of function values $f(x)$, rather than function derivatives.

To further reduce the searching complexity of the problem, an eigen-deocomposition process is employed to compress the dimensionality of the shape feature vectors. That is, we eigen-decompose the covariance matrix Σ_s in Eq. 3.2 as

$$\Sigma_s = \Phi^T \Lambda \Phi,$$

where Φ is the eigenvector matrix and Λ is the corresponding diagonal matrix of

eigenvalues. Then the Mahalanobis distance expression is converted to:

$$D = \sum_{i=1}^{2N} \frac{b_i^2}{\lambda_i},$$

where $\mathbf{b} = \mathbf{\Phi}^T \tilde{\mathbf{v}}$ is the new vector under the orthogonal transform. Due to the energy compression feature of eigen-transform, we can further simplify the above equation into

$$D = \sum_{i=1}^{k} \frac{b_i^2}{\lambda_i},$$

where $k \ll 2N$. In this sense, the original $2N$ dimension shape vector $\tilde{\mathbf{v}}$ is projected to a k dimension subspace by setting $\tilde{\mathbf{v}} = \sum_{\mathbf{i=1}}^{\mathbf{k}} \mathbf{b_i} \phi_\mathbf{i}$, where ϕ_i is the i-th eigenvector of the covariance matrix. For the same reason, our shape detection problem as defined by Eq. 3.3 is also reduced to a searching problem in a k dimension vector space.

In this work, k is chosen to be six and the downhill simplex algorithm [30] is used to find the global minimal point in this 6-dimensional subspace. On convergence of searching, the final energy term $\{\mu_i E_{\text{internal}} + \mu_e E_{\text{external}}\}$ is thresholded to determine whether a valid head-and-shoulder pattern is detected. In practice, we find this shape matching process can effectively remove the detection error introduced by simple skin color based face detection (for example, the detection errors as shown in Fig. 3-3(c)).

3.3. Temporal Analysis Module

At the system level, to further distinguish anchorperson pattern from other head-and-shoulder patterns, temporal analysis is carried out to compare the detected head-and-shoulder patterns in the temporal direction. Only those with high temporal frequency are decided to be anchorperson frames while others are discarded.

This can be realized with the following steps.

1. Generate a feature vector $f(t)$ for each detected head-and-shoulder pattern frames at the time t. This can be based on different heuristics, *i.e.*, color, texture, etc. of the head-and-shoulder regions. To accommodate certain flexibilities in the color and location of the regions, in our work, the head-and-shoulder regions are divided into uniform bins. Each bin contributes to the feature vector by its two most important colors.

2. For two head-and-shoulder patterns located at different temporal locations of $t1$ and $t2$, define their similarity (or distance) measure as:

$$|f(t1) - f(t2)|.$$

3. Using this distance measure, an unsupervised clustering algorithm [101] can be run over the feature vector set. This will cluster the feature vectors into groups. Only these groups with feature vector number above a threshold is declared as an anchorperson.

3.4. Experimental Results

The algorithm is tested on three days (03, 05, and 08 of Feb. 1999) video data from NBC Nightly News [2] in our experiments. In Fig. 3-5 to Fig. 3-8, we show some processing results. In Fig. 3-5 to Fig. 3-8, (a) is the key frame with detected face regions, (b) is the initial head-and-shoulder object contour ($\tilde{\mathbf{v}} = 0$ as defined in Eq. 3.3), and (c) is the final detected anchorperson after searching and temporal analysis. Fig 3-5 and Fig. 3-8 are two detection examples with complex backgrounds.

[2]Data was captured off the air and digitized and downscaled at AT&T Labs.

In Fig. 3-7 the anchorperson tilts his body while in Fig. 3-6 the result of face region detection is not so accurate, but after the searching process, the final contours all converge to the actual head-and-shoulder patterns.

On all the three days' data, the detection performance is $recall = 95\%$ and $precision = 91\%$ in the sense of valid head-and-shoulder pattern detection. Here *recall* is the percentage of the total number of valid patterns as judged by human beings that are actually detected by our algorithm, while *precision* is the percentage of the total number of valid patterns detected by our algorithm that are actually valid according to human judgement. After temporal analysis stage, the anchorperson detection performance is changed to: $recall = 91\%$ and $precision = 97\%$. That is, temporal analysis effectively removes fault-alarms introduced in the head-and-shoulder pattern detection stage, but it cannot remedy the miss from that stage. The major cause of detection miss, according to our observation, comes from poor results of face region detection.

(a) (b) (c)

Figure 3-5: Processing results of anchorperson detection on NBC nightly news, date 03/03/99, frame 464.

(a) (b) (c)

Figure 3-6: Processing results of anchorperson detection on NBC nightly news, date 03/05/99, frame 456.

(a) (b) (c)

Figure 3-7: Processing results of anchorperson detection on NBC nightly news, date 03/05/99, frame 497.

(a) (b) (c)

Figure 3-8: Processing results of anchorperson detection on NBC nightly news, date 03/08/99, frame 468.

3.5. Concluding Remarks

In this chapter we developed a model-based anchorperson detection algorithm for broadcast news video indexing. This model accommodates more variations in the shape, scale and location of the anchorpersons to be detected, as compared with previous reported approaches. Our experiments indicate that this algorithm work well on a large variety of frame setups. However, in order to further improve the performance, we find it is important to improve the accuracy and detection rate of the face region detector. This is part of the work that we will further investigate in the next chapter.

Chapter 4

Model Based Human Face Detection

4.1. Introduction

The detection of human faces from images and videos is also an interesting research topic for multimedia content analysis. In the past two chapters of this thesis, we studied the model designs for the segmentation and detection of head-and-shoulder patterns. In Chapter 3, in order to detect the head-and-shoulder patterns, a human face detector is designed to model the face patterns as skin-type color regions in appropriate aspect ratio. In this chapter, we investigate better ways to detect human faces.

In the literature, face detection algorithms generally come in two categories, *i.e.*, texture based approaches and color based approaches.

Texture based works try to detect face directly from the grayscale information contained in the picture. In general, most texture based methods are developed from face recognition algorithms (a good survey of face recognition and related detection algorithms can be found in [14]), because in the recognition sense, color information does not help much in distinguishing different individual's faces. Intuitively, however, it is the face texture that we human beings use to recognize different persons. Typical examples based on this observation includes the Sung and Poggio's [97]

system developed at MIT AI lab and the Rowley and Kanade [88] system developed at CMU Robotics Institute. Both systems use similar preprocessing algorithms and multi-scale searching mechanism, except that Sung and Poggio's system uses multi-modal Gaussian clustering method as classifier while Rowley and Kanade's system uses neural network as classifier. Similar systems available in the literature also include: Lew's work [56] that uses information theory, Collobert *et al.*'s [33] and McKenna's [73] works that use neural networks, Yang and Ahuja's work [113] that uses factor analyzer and Osuna *et al.*'s work [86] that uses support vector machine (SVM) as classifiers. They all reported close detection performance on some common testing pictures, for example the CMU testing database. In addition, Nefian and Hayes [83] reported the use of Hidden Markov Model (HMM) for face detection. Moghaddam and Pentland [78] suggested to use eigen-space decomposition methods for face detection purpose. However, both of the works have recognition as the purpose of their systems, their detection algorithms are only tested on some high quality mug shots that are used for recognition purpose.

In general, most texture-based algorithms are training example based. They have good detection accuracy within the scope of their training set but their missing rate climbs high for testing data outside the training data set. For example, most texture-based algorithms detect only faces in frontal views with limited tilt and rotation.

In contrast, color-based detection algorithms try to model human skin color in different chromatic spaces (RGB, YCbCr, HSV, *etc.*) with various statistical models. In the literature, typical skin color modeling works include: *mixture of Gaussian* [85], *linear region approximation* [39], *Bayesian minimal cost decision rule* [105], *etc.* In addition to color modeling, a number of recent works seek to include additional heuristics such as texture, symmetry, region ratio, region segmentation and

merging, *etc.* [5, 112, 2]. Compared with texture-based approaches, color-based algorithms are easier to design and more popular in the multimedia community. They generally have good detection rate, but suffer from high false alarm rates when the backgrounds have skin like colors.

In addition, some interesting works stand in between these two categories. Menser and Muller [76] used a skin color model to generate a *skin tone probability image*, on which they apply principle component analysis (PCA). That is, they propose to analyze the *texture* of the probability image rather than the luminance image. Wang and Chang [105] proposed to detect human faces directly on compressed MPEG macroblocks. In their work, JPEG pictures and MPEG I frames are partially decoded (entropy decoded and de-quantized) to restore DCT parameters in their block structure. Their algorithm then works directly on the decoded DCT parameters. Color information is used as the major detection clues in their algorithm. A skin color model is created at the macroblock level in the YCbCr color space. In addition, they also use some texture information by grouping the DCT parameters into bins and evaluating the energy distribution patterns based on bin statistics. In general, their approach shares the same problem with those color-based algorithms designed in the pixel domain, *i.e.*, it has a high detection rate as well as a high false alarm rate.

In this chapter, we propose a face detection algorithm that combines both color and texture information. In order to speed up the processing, we design the algorithm to work on the compressed DCT domain in a similar way as Wang's [105] work. However, our work extends theirs by building up an independent texture-based detection module in the compressed DCT domain, *i.e.*, we study how to map the successful texture-based face pattern detection algorithms such as [97, 88, 113, 86] from the traditional pixel domain to the DCT transform domain. With the new

texture-based detection model included, face detection problems can be solved more reliably in the compressed domain, as compared with Wang and Chang [105] work. We believe this algorithm is especially valuable for fast content analysis of large amount of visual media data stored in the compressed formats such as JPEG and MPEG. In the following, when mentioning compressed DCT domain, we refer to JPEG [103] pictures and MPEG I frames [38] that are partially decoded (entropy decoded and de-quantized) and have their DCT parameters restored in 8 pixel by 8 pixel block structures, if not expressed clearly otherwise.

To conform to the phrases and the structures of the whole thesis, we still refer to this approach as *model-based* face detection. Because in this work, we create statistical models for both texture-based and color-based face detection algorithms. The difference between the modeling works in this chapter and those blob models in Chapter 2, however, is that the face models used here are all example-based or data-driven models, while the blob models are semantic-based models, *i.e.*, we assume that the head-and-shoulder patterns can be represented with two blobs. Similarly, the shape model and the color model used in Chapter 3 are also example-based models.

The structure of this chapter is as follows. In Section 4.2. we discuss in detail the texture-based face detection algorithm design in the compressed DCT domain. In Section 4.3., the texture-based face model is extended to include color information and a new combined texture-based and color-based face detection algorithm is developed with experimental results. Finally in Section 4.4. we conclude the chapter.

4.2. Texture Based Face Detection in the DCT domain

In this work, we seek to map the successful texture-based face detection algorithms from the original pixel domain to the transformed DCT domain. Therefore, before

going directly to the DCT domain, we first take a close look at the available works in the pixel domain.

4.2.1 General System Structure and Complexity

In the available face detection works designed on the pixel domain, successful algorithms such as [97, 88, 113, 86] generally share the same processing procedures and structures.

In these works, the face pattern is represented as a rectangle or square window of pixel plane. In order to detect face patterns, the systems scan the input image plane at all locations. That is, the image is divided into multiple (possibly overlapping) subimages of the model window size. Each window is compared with a previously trained "face" model to tell whether it is a face pattern or not. In order to detect faces in multiple scales, the input image is downscaled to a series of scales (for example, by scales of $1.2^n, n = 1, 2, \ldots$) and the detection process is repetitively applied to each scale. Or in other words, to tell a windowed image a face or not, the window is always first scaled to the size of the model, and then compared with it.

The details of this processing flow chart are illustrated in Fig. 4-1. As we can see from Fig. 4-1, to detect faces, the input image is scaled and cropped to generate a windowed image pattern. The pattern is processed with a preprocessing module to remove illumination noise and normalize the grayscale ranges. The normalized image pattern is then fed to a classifier to see if it is a face pattern. To create a face model, a database of frontal faces is generally used. Each face is scaled and moved to align its common facial feature points such as corner of eyes, tip of nose, *etc.* to specified positions in the model window. The windowed image pattern is then processed by a preprocessor, and finally applied to train a classifier.

In all the four detection systems [97, 88, 113, 86], their basic structures are generally the same (as indicated in modules covered by the dashed rectangle in the left (Group A) of Fig. 4-1). The only different part is the classifier, *i.e.*, classifier's internal structure, its training and detection operation (as indicated in modules covered by the dashed rectangles in the right (Group B) of Fig. 4-1.

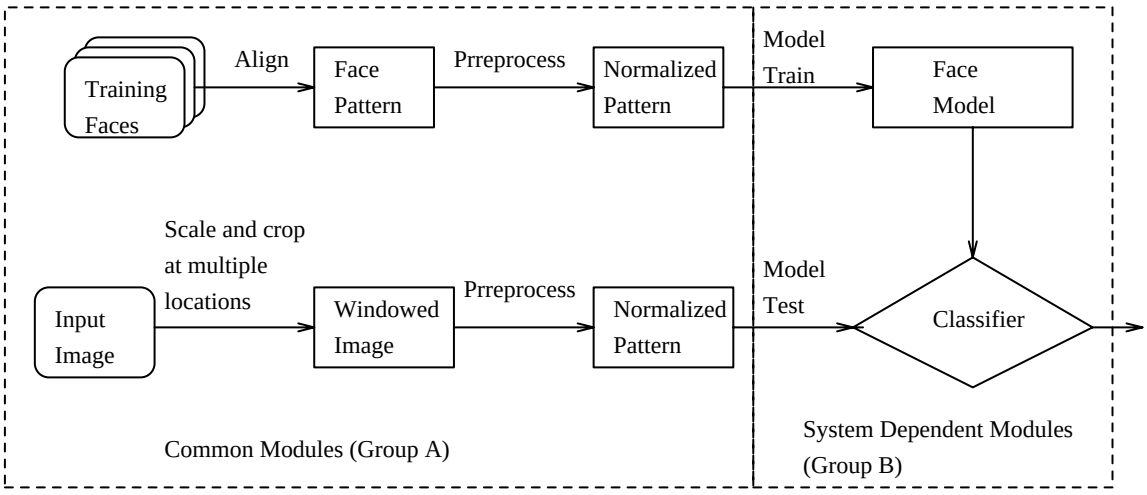


Figure 4-1: Illustration of common face detection procedures in the pixel domain

Based on the summery as depicted in Fig. 4-1, the detection complexity is M times of the processing in Group A plus the processing in Group B. M depends on the image size, the model size, and the range of the face sizes that are to be detected by the system.

However, the essence of the complexity issue is the size of the face model, which determines directly the complexity of the classifier, *i.e.*, how complex the classifier should be, in order to separate the *face patterns* from the *nonface patterns*. Though there is no clear measure available to judge the complexity of face pattern classification, most reported papers use similar face model sizes. For example, Sung [97] uses a (masked) 19-pixel by 19-pixel square window, Rowley [88] uses a (masked) 20-pixel by 20-pixel square window, and Collobert *et al.* [33] uses a 15-pixel by

25-pixel rectangle window. Therefore, the face detection problem is (at most)[1] a 2-dimensional pattern classification problem at the size of about 20 pixels by 20 pixels. If we stack up the pixels row by row as in [97], the problem is then converted to a 1-dimensional pattern classification problem in the feature space of about 200 to 400 dimension.

4.2.2 Feature Representation in the Block DCT Domain

4.2.2.1 The DCT Transform

In principle, pattern detection models should not be influenced by converting the problem to the DCT domain if we do not consider the blocks and quantization errors incurred. Because DCT is an orthonormal transform, both the *Euclid distance* and the *Mahalanobis distance* are unchanged after the transform. If we follow the Gaussian clustering approach as discussed in [97], it is easy to prove that the Gaussian model remains a Gaussian model in the DCT domain. Let's suppose the feature vectors $\mathbf{x}_i$ (in the pixel domain) have a Gaussian distribution as $N(\mathbf{x}_0, \Sigma)$, and the DCT transform matrix is denoted as $\mathbf{C}$. Then the corresponding feature vector $\mathbf{x}'$ in the DCT domain is $\mathbf{x}'_i = \mathbf{C} \cdot \mathbf{x}_i$. Because this is a linear transform, the created variants $\mathbf{x}'_i$ still have a Gaussian distribution $N(\mathbf{x}'_0, \Sigma')$, with the new mean $\mathbf{x}'_0 = \mathbf{C} \cdot \mathbf{x}_0$ and the new covariance matrix $\Sigma' = \mathbf{C} \cdot \Sigma \cdot \mathbf{C}^t$.

In addition to the invariant feature, the DCT domain is better than the pixel domain for pattern classification problems in that DCT transform reduces the dependence between individual components and compresses the feature energy to the low frequency parameters. Therefore, it is much easier to choose feature components from DCT parameters than from direct pixel values.

[1] Sung's [97] work indicates that the classification problem can be projected to subspaces in much lower dimensions. But there is no clear quantitative boundary on how low this dimension could be.

4.2.2.2 Block Quantization Problem

However, to apply model-based algorithms directly to the compressed domain of JPEG and MPEG, a major problem to overcome is that the image frames are divided into 8 by 8 blocks before DCT transform. Therefore, any detection work based on DCT parameters has to be done at the locations of blocks rather than pixels. That is, the blocks reduce the spatial resolution of the system by 8, which makes it hard to detect small faces without fully decoding the image from the block based DCT parameters to pixels. We refer to this problem as *block quantization* in this work.

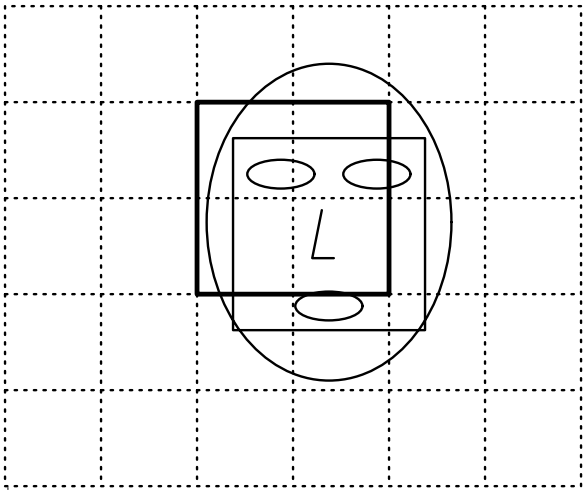


Figure 4-2: Illustration of the block quantization problem in face detection.

More specifically, the block quantization problem influences face detection in the following three aspects.

First, because the DCT parameters are organized in a block structure, in order to detect a face, a face model cannot be used to search the picture pixel by pixel. Instead, this search can only be done block by block in the DCT domain. This problem is better illustrated in Fig 4-2, in which the rectangle in light solid lines

represents the actual location of a face, while the rectangle in dark solid lines is the closest searching window that the system can arrive at based on 8 by 8 block quantization. Therefore, in order to detect faces that are not aligned with block positions, we need to introduce certain translation invariant feature in face model training. Or in other words, when training the face model, more images patterns should be included as positive training examples than the corresponding procedure that used in the pixel domain.

Second, in addition to the feature aligning problem, block quantization also introduces some background noise when the searching window is not well aligned with the actual face region, which influences the accuracy of both model training and detection.

Third, in the block-based DCT domain, it is hard to obtain resolution transformation. Though several papers [53, 77, 27] have discussed the issue of fast resolution transform in the compressed DCT domain, it is still too expensive to carry out resolution transforms in arbitrary ratios, *e.g.*, to down-sample the image by a ratio of 1.2. Therefore, in order to detection faces in multiple scales, we have to design models individually for each scale. We call this solution as a *multi-model* approach in this work, as compared to the multi-scale approach commonly used in the original pixel domain.

To sum up, the block quantization reduces the distribution density of the face patterns in the feature space, as well as introduces background noises and scaling problem. Therefore, to detect human face patterns within the block based DCT domain is more difficult than to do it in the pixel domain.

4.2.2.3 Feature Vector Design

In this work, we design face detection as a 1-dimensional vector classification problem similar to Sung's [97] system. In the DCT domain, feature vectors are created directly from (block based) DCT parameters as follows. Suppose the size of the face model is M block by M block and the desired length of the feature vector is N, then in each DCT block, the lowest d DCT parameters are used for feature vector creation, where

$$d = N/(M * M).$$

This is better illustrated in Fig. 4-3, in which the lowest d DCT parameters from each block is stacked up to form a N dimension feature vector.

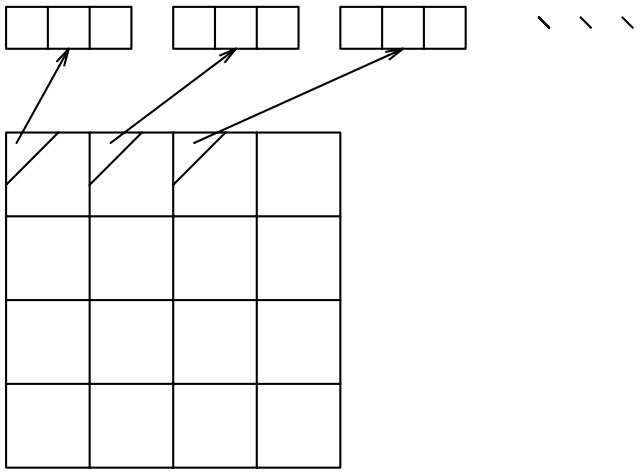


Figure 4-3: Illustration of the feature vector creation from DCT parameters

In order to choose the first few low frequencies from a 2-D DCT block, we number the 64 DCT parameters according to the typical DCT quantization table design as reported in [99], *i.e.*, the larger the quantizer is, the less important its corresponding DCT parameter is for low frequency representation of the picture. In Eq.4.1, we show the positions of the first 16 parameters.

$$\begin{pmatrix} 1 & 2 & 3 & 10 \\ 4 & 5 & 7 & 13 \\ 8 & 6 & 11 & 15 \\ 9 & 12 & 14 & 16 \end{pmatrix}. \tag{4.1}$$

Based on the complexity analysis in Section 4.2.1, the complexity of face detection problem should be processed adequately over a rectangular window of 19x19 pixels, therefore, the feature vector length in this work should be in the range of 200 to 400. For example, if a 5 block by 5 block face model is to be created, and we choose the feature vector length to be 300, then for each DCT block, its lowest 12 DCT parameters are used to create the feature vector for training and classification.

In principle, this approach of choosing the lowest few DCT parameters in each block is actually to downscale the windowed image to a *unit* model size, saying 19 pixel by 19 pixel, as used in the pixel domain. In other words, to tell if a windowed image pattern is a face or not, in the pixel domain, the face is downscaled to the unit model size, *e.g.*, 19 by 19 pixel, and compared with the face model. In the DCT domain, the correspondence is that the low frequency DCT parameters in each block of the windowed image pattern are used to form a feature vector, which is then compared with the face model. In this mapping, as long as the lowest few DCT parameters have maintained the image features at the resolution of 19 by 19 pixels (approximately)[2], we have reason to believe that those parameters have maintained the necessary information for face detection, based on our complexity analysis in Section 4.2.1. The benefit in this domain mapping is that unlike the bilinear downsampling in the pixel domain, choosing the lowest few DCT parameters

[2]That is, if the image is decoded with only these low frequent parameters and then downscaled to 19 by 19 pixels, the image quality is still comparable with direct downloading the original image in the pixel domain.

is an easier and better way to do downsampling in the DCT domain. This is also noticed and discussed by Dugad and Ahuja [27] in their paper on fast DCT domain downsampling design.

To better illustrate this problem, we design a simple experiment on Lena image (256 pixel by 256 pixel, grayscale). In the DCT domain, we maintain only the first N low frequencies (according to the sequence defined in Eq.4.1) in each DCT block and truncate the rest into zero. The truncated image is then converted back into the pixel domain and its restored quality is compared with the original image by PSNR. In the pixel domain, we first downscale the image into a smaller size of M by M, and then upscale it back to the original size of 256 pixel by 256 pixel with bilinear interpolation. After that, the restored image is compared with the original image with PSNR. Here these two processes are two different approaches to do downscaling and upscaling. If we impose that the feature vector of the downscaled image in both domains should be the same[3], then the parameter M and N should have the following relation:

$$N = 32 \cdot \sqrt{M}.$$

In our experiment, the feature vector length is changed from 1 to 16 per DCT block, and the corresponding PSNR data for both pixel and DCT domain is shown in Fig.4-4. We could see from the figure that given the limitation of the same feature vector length, the corresponding downscaled image generated in the DCT domain has better PSNR quality than that generated in the pixel domain. One straightforward explanation for this difference is the energy compression feature of the DCT transformation.

[3]We assume the feature vector in the pixel domain is obtained by stacking up the pixel values row by row, as is defined by Sung in [97].

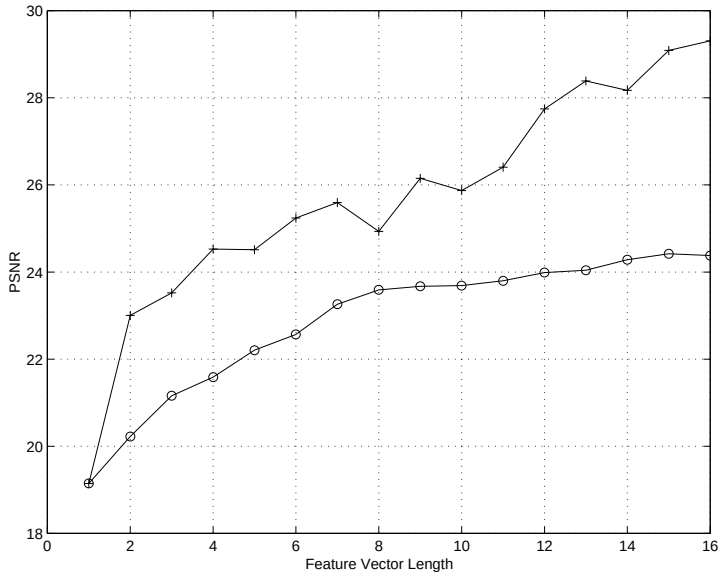


Figure 4-4: Comparison of the image downscaling performance in the pixel domain and in the DCT domain (with Lena image). The "+" curve is the DCT domain data and the "o" curve is the pixel domain data.

In addition, it is also worth noticing that the DCT curve in Fig.4-4 is not strictly convex, which indicates that a simple truncation of the high frequent parameters is not the best approximation of the image in the DCT domain (certain filters may improve the quality). However, as compared with the performance curve defined in the pixel domain, the truncation-based approach is much better.

4.2.3 Face Detection System Design

Based on the analysis in Section 4.2.1, a multi-model DCT domain face detection system is designed that combines algorithm capabilities and performance efficiencies.

4.2.3.1 Multi-Model Detection System

Due to the block quantization problem as discussed in Section 4.2.2.2, model-based face detection in the block-based DCT domain faces two problems, *i.e.*, aligning problem and scaling problem.

For the aligning problem, because the block size is 8 pixels, there are totally 64 possible spatial setups at the every searching position in the DCT domain. This is depicted in Fig. 4-5, in which the square in dashed lines is the ideal cropping window based on face feature aligning, the square in light solid lines is the model window at the closest searching position based on DCT block structure. The square in dark solid lines is an 8-pixel by 8-pixel block, in which the top-left corner of the dashed window has to lie. There are, therefore, 64 possible positions. To solve the aligning problem, one might train 64 models for one model size, with each one representing a face pattern at a different aligning position.

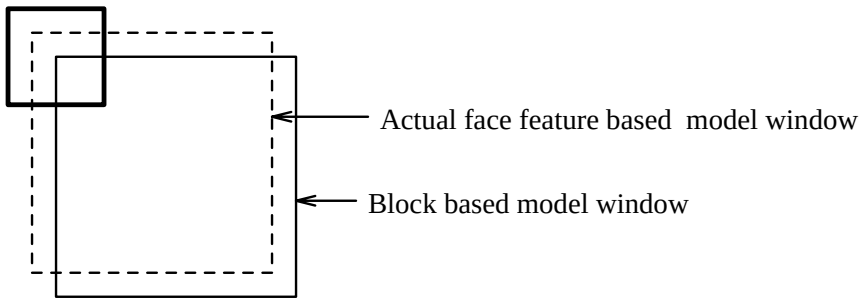


Figure 4-5: Illustration of possible different spatial relations between the actual face feature based model window and the DCT block based model window.

However, this approach is obviously not efficient because 64 models are hard to store as well as to apply. In addition, there are redundancies in the 64 models as the spatial neighboring models are similar to each other. Therefore, a trade-off has to be made between model efficiency and model accuracy. In addition, to overcome the scaling problem in the DCT domain, multiple models have to be created in order to detect faces in different scales, which further improves the burden of choosing too many models in one scale.

In this system, we create face models in six scales, and the model windows are designed to be squares in side length of 5, 6, 7, 8, 9, and 10 DCT blocks. That is,

faces in the range of 40 by 40 pixels to 80 by 80 pixels are covered by the system. For each scale, one face model is trained for faces in all the possible aligning positions, *i.e.* the model represents variations in different face features as well as in different aligning positions (all the 64 possible positions).

The processing procedures for each model size is generally the same as those in the pixel domain (refer to Fig. 4-1), except that the works are moved to the DCT domain. We illustrate the DCT domain detection procedures and model training procedures separately in Fig. 4-6 and Fig. 4-7. The details of the processing modules are discussed in the following sections.

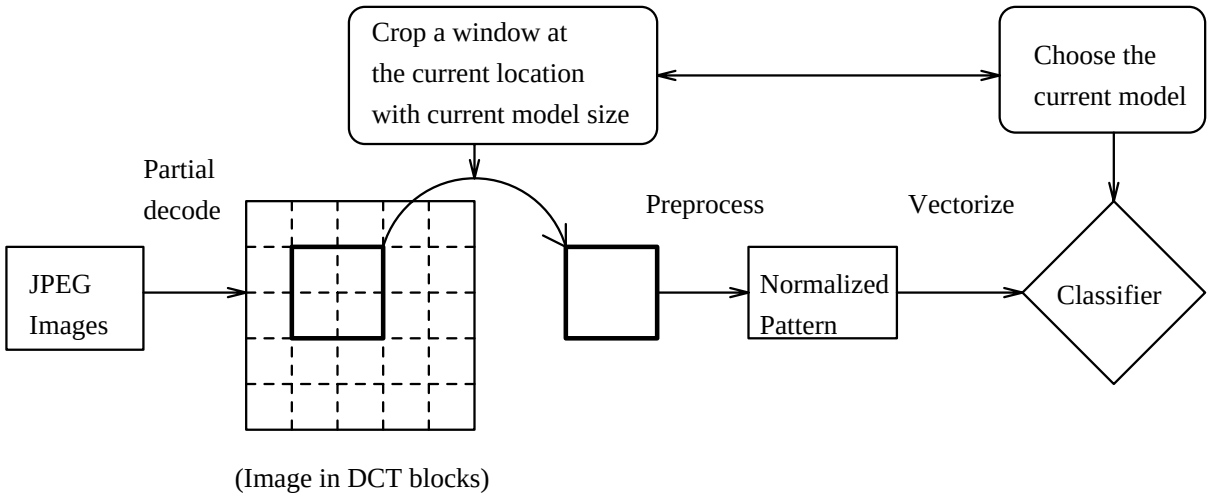


Figure 4-6: Illustration of the face detection procedures in the DCT domain.

4.2.3.2 Preprocessing and Masking in the DCT Domain

To remove the signal variance introduced by different illuminations and different grayscale dynamic range, a number of preprocessing steps are used in the works designed in the pixel domain. In the DCT domain, we find it also possible to implement their correspondences. More specifically, our system includes the following

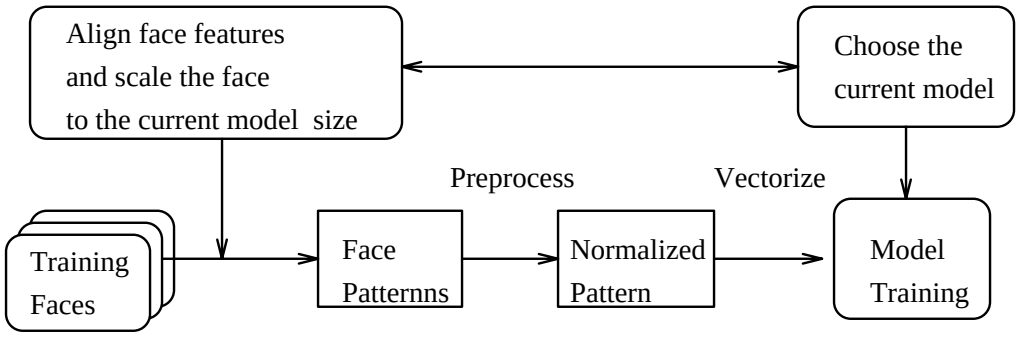


Figure 4-7: Illustration of the face model training procedures in the DCT domain.

preprocessing steps:

1. **Masking.** Similar to Sung's [97] work, we introduce binary face masks on top of DCT blocks. For face patterns, these masked blocks often represent background regions. Removing them from the feature vectors ensures that the subsequent modeling work does not wrongly encode any unwanted background structures. Based on different sizes of the face model, different masks are designed to represent different spatial resolutions. In Fig. 4-9, we show the six masks we used in our system for face models in six different scales. Note that the face models are actually in different sizes. They are scaled to the same size in Fig. 4-9 for ease of illustration. Each block in the model windows is of size 8 pixel by 8 pixel.

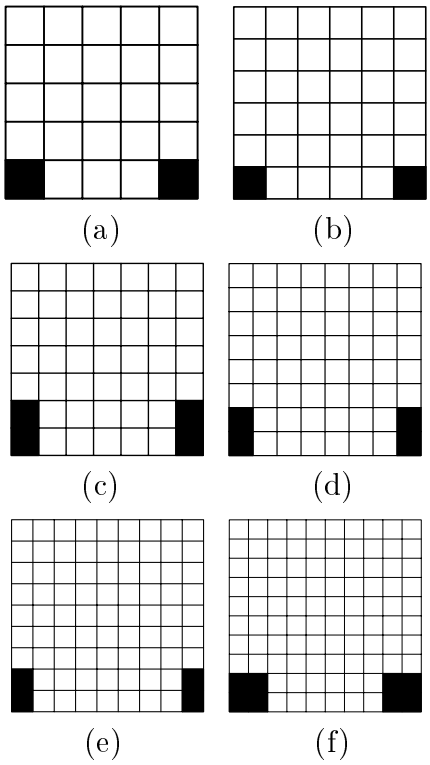
2. **Illumination Linear Factor Correction.** In order to remove shadowing effect, a 2-dimensional linear function is fitted to the DC plane of the face region in the DCT domain. The fitted function is then removed from the DC plane.

3. **Histogram Equalization.** This nonlinear process is applied directly to the DC components of the face region DCT blocks. In addition, the AC components in each block are changed linearly to reduce the block effects. That is,

for each DCT block, if its DC component d is mapped to d' according to the histogram equalization, then its AC components a_i are also mapped linearly to a_i' with $a_i' = a_i * d'/d$.

(a) (b) (c)

Figure 4-8: An example of preprocessing results on block based DCT domain



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

Figure 4-9: Illustration of binary face masks design for models in different sizes.

Fig. 4-8 shows an example of preprocessing (the picture is taken from Olivetti face database[4]). Fig. 4-8(a) is a cropped face region, (b) is the face region with a grayscale linear factor removed, and (c) is the final result of face region preprocessing. We can see that the shading in the original face is effectively removed (in subfigure (b)). In addition, the histogram equalization based on DC components introduces certain blocking effect, but the general quality is acceptable (in subfigure (c)).

4.2.3.3 Distribution Based Classifier Design

For face detection purpose, a number of classifiers have been used. These include: unimodel Gaussian [78], multimodel Gaussian [97], neural networks [88, 73] and support vector machine (SVM) [86]. Among them, neural networks and support vector machine have been shown to have theoretical merits for high dimensional vector classification problems. Especially SVM is proved capable of finding optimal classification boundary that minimizes structural risk [9]. However, both neural networks and SVM are hard to train, *i.e.*, to organize the positive and negative training samples in order to make the classifier converge to the optimal status.

In this work, we have to train multiple face models for faces in multiple scales, therefore, we choose to use multimodal Gaussian model as classifier, which is a trade-off between model training complexity and classification performance.

The essence of multimodal Gaussian model is to approximate the feature vector distribution with a number of Gaussian clusters. Though Gaussian distribution is a general purpose statistical model that has been extensively used, unimodel Gaussian distribution is shown by Sung [97] inadequate to model face patterns' distribution in high dimensional feature space. Multimodal Gaussian model is a natural extension of unimodal Gaussian models that has worked well in complex and

[4] `http://www.cam-orl.co.uk/facedatabase.html`

high dimensional distribution problems. It also has moderate training complexity as compared with neural networks and SVMs. In this work, in order to improve classification performance, we use two groups of multimodal Gaussian models to approximate separately the distribution of face patterns and nonface patterns.

Positive Training Samples The basic idea of this approach is to approximate the distribution of face feature vectors with high dimensional Gaussian clusters. To study the distribution of face patterns in the DCT domain, 300 frontal faces are collected from various sources, such as MIT face database, Yale face database, Olivetti face database, University of Sterling face database[5] and some anchorperson images from the NBC news video database stored at AT&T Research Labs. Four feature points of each face, *i.e.*, the inner and outer corners of both eyes, are marked manually. Each face is moved and scaled to align these feature points with specific positions in a model window. The face image is then cropped with the model window, and converted to the DCT domain. After that, the feature vector is obtained from the DCT parameters as described in Section 4.2.2.3. Because we train only one model for one modeling window size, feature vectors should also be created to represent face patterns whose feature points are not perfectly aligned with the face model. In this work, we use 16 out of 64 possible spatial aligning positions for each training face to generate training samples. In addition, to make use of the symmetric property, each training face is flipped and used as a new training sample. Therefore, in this system, totally $16 * 2 * 300 = 9600$ feature vectors are used as positive training samples.

Clustering Algorithm We cluster the face feature vectors into six clusters of Gaussian distribution. The clustering algorithm used here is similar to K-means

[5]Most of them are downloadable from `http://www.cs.rug.nl/~peterkr/FACE/face.html`

algorithm, except that the *Euclidean distance measure* is replaced by a *logarithmic Gaussian distance measure*. If we denote each cluster with a Gaussian distribution $\mathcal{G}(\mathbf{v}_i, \mathbf{C}_i)$. A new feature vector's distance to each cluster is defined to be a *logarithmic Gaussian distance* as:

$$d = \frac{1}{2}(N \ln 2\pi + \ln |\mathbf{C}_i| + (\mathbf{v} - \mathbf{v}_i)^T \mathbf{C}_i^{-1}(\mathbf{v} - \mathbf{v}_i)),$$

where N is the dimension of the feature vectors. Because N is a rather high dimension in this problem ($200 \sim 400$), we actually use Karhunen-Loeve transform to reduce the above equation into a lower dimension problem as:

$$d = \frac{1}{2}\left[M \ln 2\pi + \sum_{k=0}^{M-1} \ln |\lambda_k| + \sum_{k=0}^{M-1} \frac{y_k^2}{\lambda_k}\right],$$

where y_k^2 are the principle components and λ_k are the corresponding eigenvalues. M is the number of eigenvectors used to approximate the system. Generally we have $M \ll N$.

The detailed clustering steps are as follows.

1. Initialize the clustering process by grouping the feature vectors into six groups in *Euclidean* distance space, *i.e.*, a vector is put into the group whose center is the closest to it among the six groups. And the covariance matrix for each cluster is initialized to be unit matrix.

2. Re-compute the data centers of each cluster to be the center of the current cluster partition.

3. Based on the current cluster centers and covariance matrixes, re-assign the data partition by assigning the feature vectors to the cluster that is closest to it in the *logarithmic Gaussian distance* space. If the difference between the

new data partition and the old one is bigger than a threshold and the inner loop time (Step 2 and Step 3) is less than the maximal time, goto Step 2, otherwise goto Step 4.

4. Re-compute the covariance matrixes of the 6 clusters based on the current data partition.

5. Based on the current cluster centers and covariance matrixes, re-assign the data partition by assigning the feature vectors to the cluster that is closest to it in the *logarithmic Gaussian distance* space. If the difference between the new data partition and the old one is bigger than a threshold and the outer loop time (Step 2 to Step 5) is less than the maximal time, goto Step 2. Otherwise, return the created mean vector and covariance matrix of each cluster.

In Fig. 4-10, we give the clustering results on the 9600 face feature vectors. The model size used is 5 blocks by 5 blocks. Fig. 4-10(a) to (f) are the average faces of the six clusters at the time of convergence. Because only the 4 lowest frequent DCT parameters (the choice of 4 will be discussed later in this section) in each block are used for feature vector creation, the block effect is noticeable. But in general the six mean faces represent mainly the variances in face textures rather than those in different spatial aligning positions.

Negative Training Samples In our experiments, we find many face like "non-face" patterns in our testing examples that can not be simply separated from face patterns by thresholding. In order to reduce the misclassification rate, we further create a multimodal Gaussian model for these face like negative patterns.

The training samples are collected in a *boot-strap* fashion. That is, the positive face model is first created by the clustering algorithm as previously discussed. Based

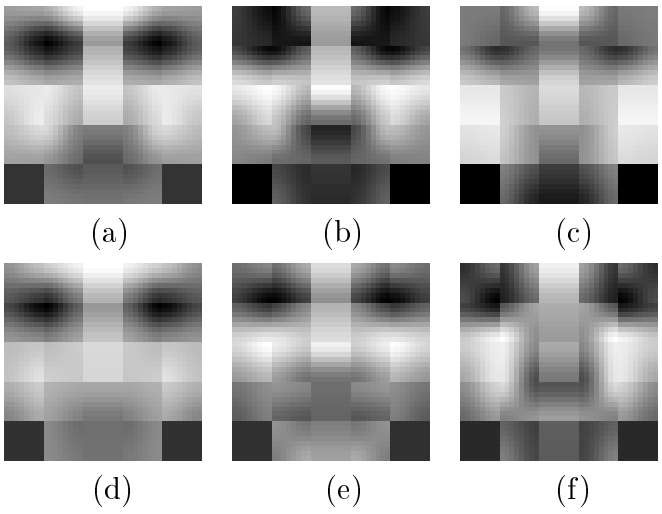


Figure 4-10: *Average faces* of the six face clusters when the face model size is 5 blocks by 5 blocks.

on this model, a face detector is designed, which is then applied to the training pictures. The nonface patterns misclassified as faces by the face detector are used as negative samples.

In this way, we collect about 9000 negative nonface samples and cluster them into eight clusters. The clustering algorithm used is the same as positive face sample clustering. The clustering result of model size 5 blocks by 5 blocks is shown in Fig. 4-11, in which each subfigure represents an average nonface pattern for the eight clusters.

Classification The classification problem is based on the distance measures from the input feature vector to the positive and the negative clusters. Let's denote the input vector as $\mathbf{v}$, the six positive clusters as $N(\mathbf{C}_k^{(p)}, \mathbf{v}_k^{(p)})$, $k = 1, 2, \cdots, 6$, and the eight negative clusters as $N(\mathbf{C}_k^{(n)}, \mathbf{v}_k^{(n)})$, $k = 1, 2, \cdots, 8$. Then a 6-dimension positive distance vector could be defined as $\mathbf{d}^{(p)} = (d_1^{(p)}, d_2^{(p)}, \cdots, d_6^{(p)})$, where

$$d_k^{(p)} = \frac{1}{2}(N \ln 2\pi + \ln |\mathbf{C}| + (\mathbf{v} - \mathbf{v}_k^{(p)})^T \mathbf{C}^{-1}(\mathbf{v} - \mathbf{v}_k^{(p)})), \tag{4.2}$$

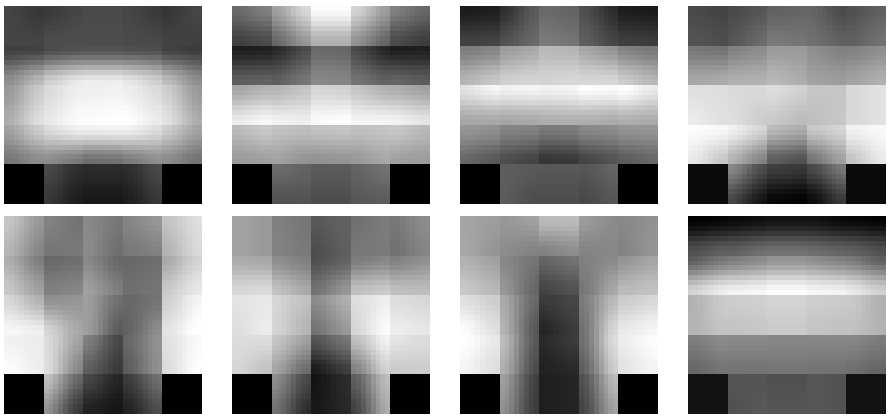


Figure 4-11: *Average nonfaces* of the eight nonface clusters when the face model size is 5 blocks by 5 blocks

where N is the dimension of the feature vector $\mathbf{v}$. In practice, because the input feature vectors are in high dimension, the covariance matrix $\mathbf{C}$ is always decomposed with KL transform:

$$\mathbf{C} = \mathbf{T}\mathbf{\Lambda}\mathbf{T}^{-1},$$

where $\mathbf{T}$ is the eigen-matrix and $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues. With only the first M eigenvectors of the eigen-matrix $\mathbf{T}$ used to span the feature space, the distance Eq. 4.2 is decomposed into two parts: the distance in the feature space (DIFS) and the distance from the feature space (DFFS),

$$\text{DIFS} = \frac{1}{2}\left[M \ln 2\pi + \sum_{k=0}^{M-1} \ln|\lambda_k| + \sum_{k=0}^{M-1} \frac{y_k^2}{\lambda_k}\right],$$

$$\text{DFFS} = \frac{1}{2}\left[(N-M)\ln 2\pi + (\ln|C| - \sum_{k=0}^{M-1} \ln|\lambda_k|) + \frac{\Delta^2}{\rho}\right],$$

where y_k is the principle components and $\Delta^2 = ||\mathbf{v}|| - \sum_{k=0}^{M-1} y_k^2$ is the residue. ρ is a weighting factor based on the estimation of eigenvalues $\lambda_k, (k = M, M+1, \cdots, N)$.

In this work, the distance measure is therefore defined as:

$$d_k^{(p)} = \frac{1}{2}\left[N \ln 2\pi + \ln|\mathbf{C}| + \sum_{k=0}^{M-1} \ln|\lambda_k| + \eta\frac{\Delta^2}{\lambda_M}\right], \tag{4.3}$$

where η is an adjustable weighting factor.

Similarly, an 8-dimension negative distance vector is defined as $\mathbf{d^{(n)}} = (d_1^{(n)}, d_2^{(n)}, \cdots, d_8^{(n)})$, with respect to the eight negative clusters.

Therefore, the classification problem is reduced from a high dimension problem ($N = 200 \sim 400$) to a lower dimension problem with $N = 14$, which is then solved with a simple minimal distance classification algorithm in our work, *i.e.*, if

$$\min_{k=(1,\cdots,6)} d_k^{(p)} \leq \min_{k=(1,\cdots,8)} d_k^{(n)},$$

the pattern is detected as a face, otherwise it is a nonface.

In implementing the classifier on practical face images, we tried to select feature vectors of a variety of lengths. Based on the complexity analysis in Section 4.2.1, the complexity of the face detection problem requires feature vectors of length in the range of 200-400 dimensions. However, we notice that when feature vectors coming from various spatial aligning positions are included as positive training samples (due to the block quantization problem), the high frequency DCT parameters become unstable in both face model training and face detection functions. In order to illustrate this problem, an experiment is carried out to measure the *separability* feature between the 9600 positive samples and the 9000 negative samples under the condition of different feature vector lengths.

The measure we used here is the *divergence* measure as defined in [37]. The divergence measure of two Gaussian distributions $N(\mathbf{v}_1, \mathbf{C}_1)$ and $N(\mathbf{v}_2, \mathbf{C}_2)$ is given

as

$$\text{Div} = \frac{1}{2}(\mathbf{v}_1 - \mathbf{v}_2)^T(\mathbf{C}_1^{-1} + \mathbf{C}_2^{-1})(\mathbf{v}_1 - \mathbf{v}_2) + \frac{1}{2}\text{tr}[(\mathbf{C}_1 - \mathbf{C}_2)(\mathbf{C}_2^{-1} - \mathbf{C}_1^{-1})] \quad (4.4)$$

To study the influence of the feature vector length on the separability of the problem, both the 9600 positive samples and the 9000 negative samples are projected to the six positive clusters, which generates two groups of feature vectors in Gaussian distribution (in $\mathcal{R}^6$). Their divergence is then computed according to Eq. 4.4. In this experiment, we used a model size of 5 blocks by 5 blocks (40 by 40 pixels). The feature vector length is changed from 23 to 230, or 1 parameter per DCT block to 10 parameters per DCT block. The corresponding divergence is computed and their relation is illustrated in Fig. 4-12.

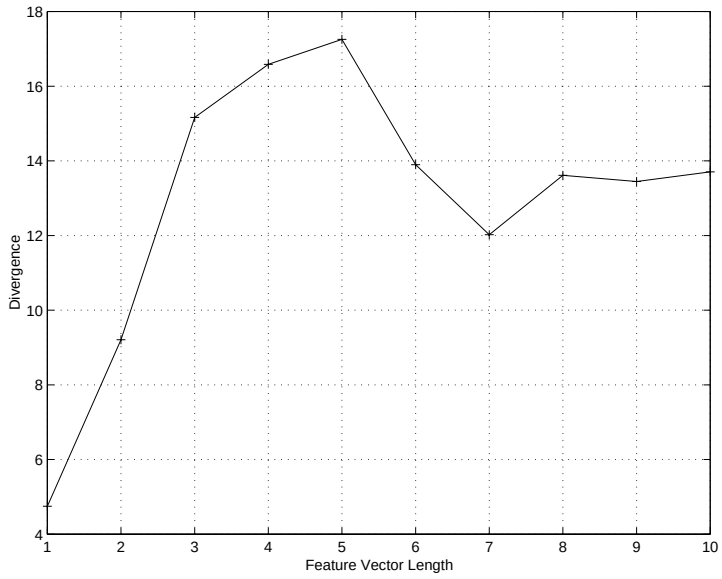


Figure 4-12: Illustration of the relation between the feature vector length and the divergence of the face and nonface patterns.

From Fig. 4-12, we notice that the divergence increases with the feature vector length from 1 to 5 (parameters per block), and then drops when the feature vector

	model size (in block)	model size (in pixel)	masked block numbers	parameters per block	feature vector length
1	5	40	23	4	92
2	6	48	32	3	102
3	7	56	45	2	90
4	8	64	60	2	120
5	9	72	77	2	154
6	10	80	92	1	92

Table 4.1: Feature vector length at different model sizes

length further increases. That is, the more DCT parameters are included into the feature vector, the less likely that the face patterns are able to be separated from the nonface ones. This problem comes mainly from the high frequency components in each DCT blocks. When we try to build up one model to represent face patterns in all the 64 different spatial aligning positions (with respect to the current model window), the high frequent parameters in the feature vectors experience large variation from sample to sample, which makes them contribute negatively to the separation problem. As indicated by Fig 4-12, for the specific case of model size 5 block by 5 block, the ideal feature vector length should be 4 to 5 DCT parameters per block, or of about 100 parameters for the entire feature vector.

Based on this observation, we determine the feature vector lengths for all the 6 models in our system, which is listed in Table. 4.1. Note that the feature vector lengths are all shorter than those used for pixel domain detection. This difference is mainly, as have already been pointed out, due to the block quantization problem in the DCT domain.

4.2.4 Experiments

The texture based face detection algorithm is tested on a variety of pictures, which include: CMU database[6], CMU online face database[7], key frames of news video clips from CNN and NBC as well as pictures downloaded from Internet and scanned from magazines and books. Though coming from different sources and formats, the pictures are all compressed into JPEG format with Adobe Photoshop 5.0 (quality option *medium*, quantizer index 3) before processed. Some detection results are shown in Fig 4-13.

Compared with pixel-domain detection algorithms, our algorithm is less accurate because of the block quantization issue, especially the false detection rate is relatively high. This can be seen from Fig 4-13. To overcome this problem, we seek to combine the texture-based algorithm with a color-based algorithm.

4.3. Combined Color-Based and Texture-Based Face Detection in the DCT Domain

4.3.1 Generating Color Similarity Map

Color-based face detection work in the DCT domain was first discussed by Wang and Chang [105]. In this work, our basic design is similar to theirs, except that we do not try to setup a threshold at the color detection stage.

We assume the color pictures in JPEG and MPEG I frames under study are compressed in $4:2:0$ format. The pictures are partially decoded to restore their DCT parameters in the block structures. The skin color is modeled and detected at the macroblock level. That is, in each macroblock, the DC components of the Cb

[6] `http://www.ius.cs.cmu.edu/IUS/eyes_usr17/har/har1/usr0/har/faces/test/`

[7] `http://www.ius.cs.cmu.edu/IUS/usrp0/har/FaceDemo/gallery-inline.html`

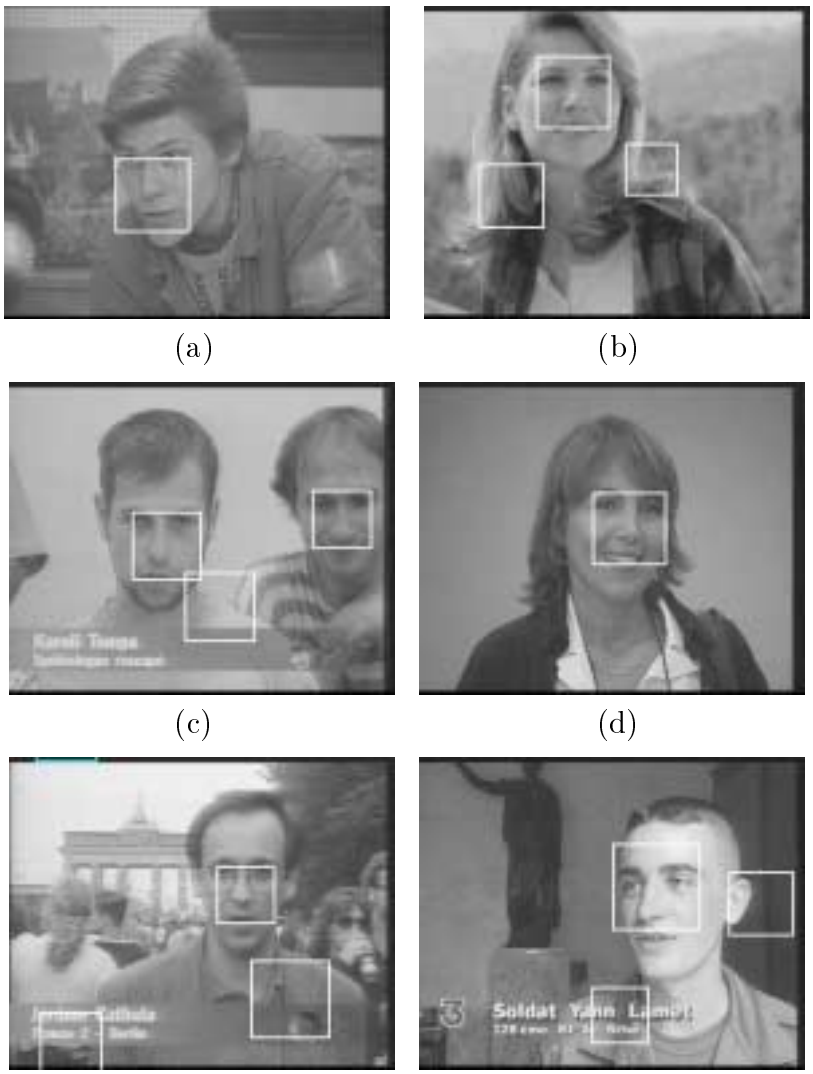


Figure 4-13: Several detection results of our texture-based face detection algorithm (in the compressed domain)

and Cr blocks are used as the average chromatic feature vector. A Gaussian model $N(\mathbf{v}_s, \mathbf{C}_s)$ for the skin color is created by training over a manually labeled picture database. Based on this skin color model, a color similarity map is generated for each picture at the macroblock level. For macroblock (i, j), if we denote its color feature vector as $\mathbf{v}_{(i,j)}$, then its skin color similarity map entry is

$$\text{color}(i,j) = -\frac{1}{2}\left[\ln 2\pi + \ln|\mathbf{C}_s| + (\mathbf{v}_{(i,j)} - \mathbf{v}_s)^T \mathbf{C}_s^{-1}(\mathbf{v}_{(i,j)} - \mathbf{v}_s)\right] \tag{4.5}$$

In this way, a skin-color similarity map is created for an input picture at the macroblock level.

4.3.2 Color Constrained Face Pattern Detection

With the color map available, the face detection problem is further extended from previous texture domain to the color domain. That is, given an input color picture in its YCbCr format, we can apply the texture-based pattern detection on the Y component and the skin-color map based pattern detection on the CbCr components[8]. Because both detection designs have a statistical expression, it is easy to combine them with a statistical framework. In other words, the color-based and texture-based detection modules could be designed to work independently of each other and they processing results could be combined statistically for the system to generate the final results. This processing structure is shown in Fig. 4-14, which we refer to as parallel merging structure.

In contrast, an alternative approach is sequential merging structure. We show its flowchart in Fig. 4-15. Though theoretically the parallel structure has the merit

[8]Because the skin-color similarity map is a map of scalar values, it is straight-forward to design a face pattern detection algorithm based on this similarity map, which should be, in principle, the same as the one we have designed on the texture map.

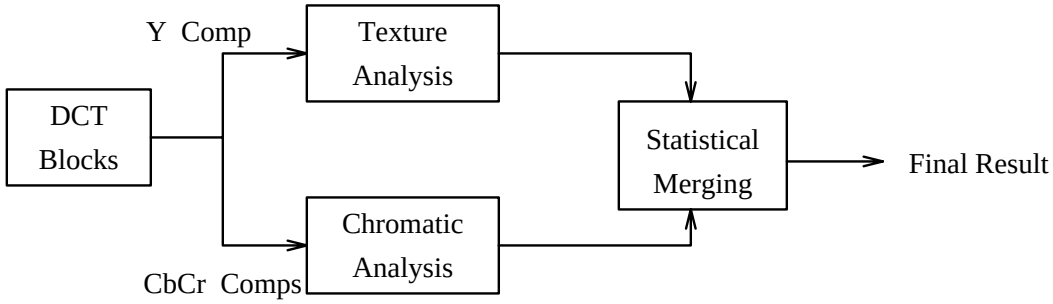


Figure 4-14: Parallel merging structure for combined face detection system

of delayed judgment and thus better detection performance, sequential structure is simpler and faster. In addition, the chromatic detection module is not a balanced module as compared with the texture detection module in the detection accuracy and reliability sense[9]. Therefore we follow the sequential structure in our system design. As indicated in Fig.4-15, a simple thresholding function is applied right after the color analysis module to generate a skin color map as follows. Given a windowed image pattern $\mathcal{W}$, its average skin-color similarity

$$\frac{\sum_{(i,j)\in\mathcal{W}} \text{color}(i,j)}{\sum_{(i,j)\in\mathcal{W}} 1},$$

(where $color(i,j)$ is defined in Eq. 4.5), is compared with a threshold T. Only the windows with an average similarity higher than the threshold are further processed by the texture analysis module as discussed in Section 4.2..

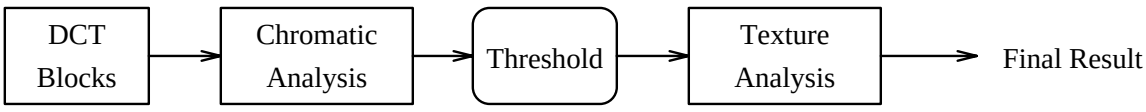


Figure 4-15: Sequential merging structure for combined face detection system

This combination, though simple, is better than both texture-only and color-

[9]In this work, we do not spend time to create an example based face pattern on top of the skin-color similarity map as we do in Section 4.2. on top of the texture map (though it is possible).

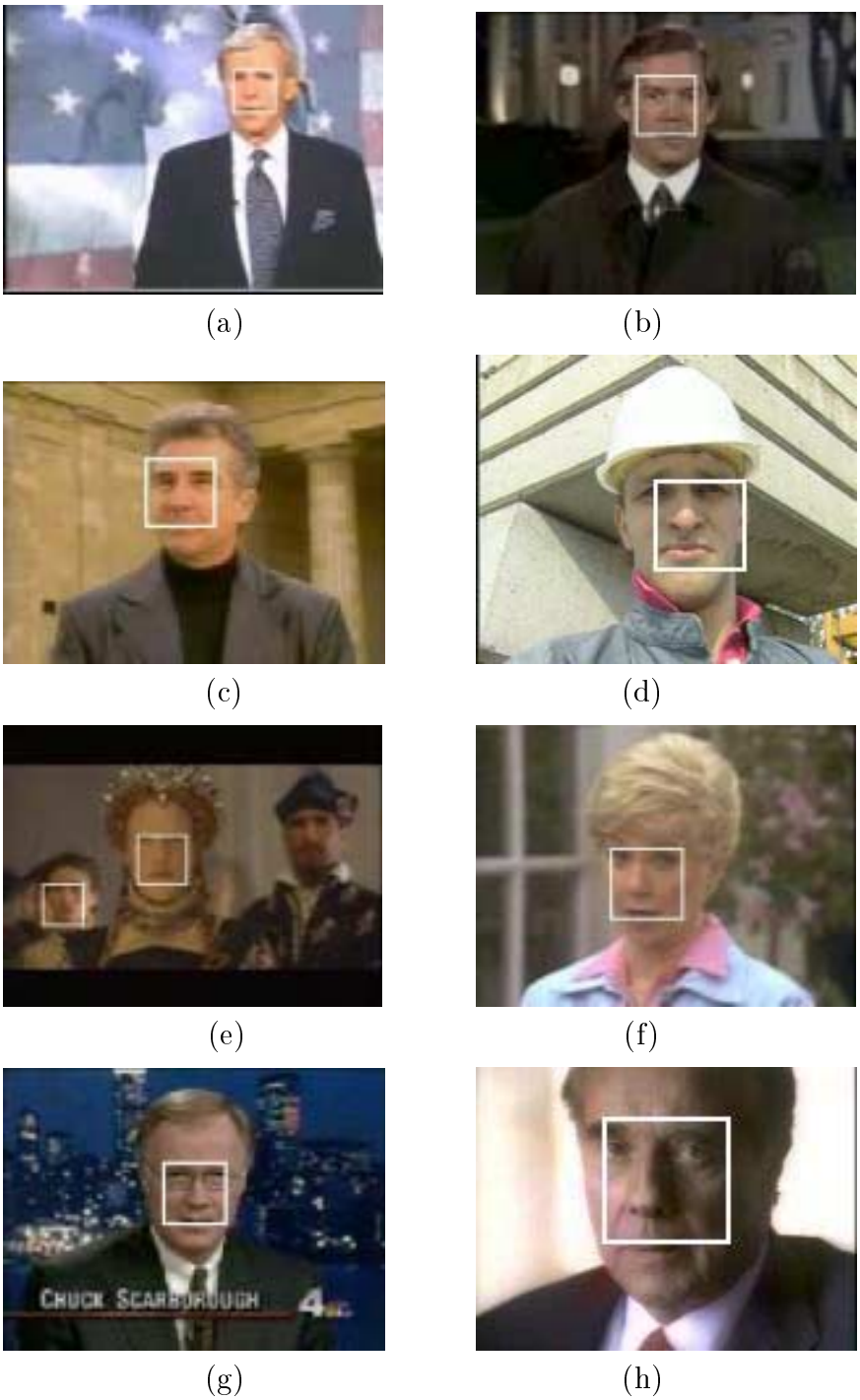
only algorithms. Compared with texture-only algorithms, the color similarity map offers an additional constraint, which eliminates false alarms in the background without a skin-color appearance. Compared with color-only algorithms, the texture analysis module helps reduce false alarms introduced by skin-like backgrounds. In addition, the texture analysis module is also useful in locating faces when skin-like backgrounds are close to actual faces, in which case the simple shape analysis modules commonly used in the color-based algorithms always fail.

4.3.3 Experiments

We tested our face detection algorithm based on combined texture and color information over many pictures. The testing picture set we used in Section 4.2.4 is also used here, except the CMU database and CMU online database are not used because they are grayscale pictures. In Fig. 4-16 and Fig 4-17, we show some detection results on pictures from various sources.

To evaluate the performance of our algorithm quantitatively, we use the key frames of one day's NBC Nightly News (Feb. 18, 1999, totally 586 frames) as our testing data set. Within this data set, there are 42 faces and 36 of them are frontal and upright ones that are within the coverage of our texture-based face model. The detection performance is listed in Table 4.2. In Table 4.2, the new combined texture and color based face detection algorithm is compared with the color-based detection algorithm that we discussed in Section 3.2.2. Because the color-based algorithm works in the pixel domain, while the combined texture and color based detection algorithm works in the compressed DCT domain, the comparison is focused on the detection performance, but not the spatial location accuracy of the detected faces.

As we can see from Table 4.2, the combined texture and color based algorithm has less false alarms than the color-based algorithm because the texture processing



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 4-16: Face detection examples

(a) (b)

Figure 4-17: Face detection examples, continued

	color based detection	color & texture based detection
total faces	42	36
face detected	34	30
face missed	7	5
false alarm	9	4
detection rate	81%	83%
detection accuracy	79%	88%

Table 4.2: Performance of the combined color and texture based face detection algorithm

module has removed some false alarms introduced by the skin-color backgrounds. In addition, less faces are missed by the combined color and texture based algorithm than by the color-based algorithm. This difference is mainly due to the different ability of the two algorithms to detect faces surrounded by skin-like backgrounds. That is, in color-based algorithms, skin-color regions are first detected with a skin-color model based detector and then processed with a shape analysis module. Only the regions have specific aspect ratios are accepted as face regions. When some skin-like backgrounds are close to the actual faces, the detected skin-color regions always get connected, which makes the shape of the detected regions no longer have a face-like shape. In contrast, the texture based approach is not influenced by the backgrounds, as long as they do not *look like* face patterns in the grayscale sense.

As a result, the combined texture and color based algorithm exhibits better *detection rate* (*i.e.*, the ratio of correctly detected faces v.s. total faces that should be detected) as well as better *detection accuracy* (*i.e.*, the ratio of correctly detected faces v.s. the total detected faces) in this experiment.

4.4. Concluding Remarks

In this chapter we mainly developed a texture-based face detection algorithm that works in the compressed DCT domain. This is a new work on compressed domain

processing. Our work is based on the previous face detection works designed in the pixel domain, but we discussed major problems we met in the compressed DCT domain, such as block quantization problem, feature vector selection, preprocessing design in the DCT domain, and multi-model based system structure. Due to the block quantization problem, we have to use shorter feature vector than the face detection designs in the pixel domain. Therefore, the proposed texture-based detection algorithm is not as good as its counterparts in the pixel domain. To solve this problem, we proposed to combine the texture-based algorithm with the face color detection algorithm we discussed in Chapter 3. Experiments indicate that the combined texture and color based detection algorithm works better than both texture-only and color-only algorithms.

To sum up, this work is interesting because it first proposed to do traditional pattern detection work on the compressed DCT domain, which is a promising research direction for multimedia content analysis. In addition, in this work we also proposed, for the first time, to combine texture and color information for face detection. This is especially useful for multimedia processing, in which most visual data are in color formats.

However, several problems in this work still need further study in the near future.

First, in this work, we simply choose to represent faces in one model size with one model. Therefore, the model has to represent variations in different faces as well as variations in different spatial aligning due to the block quantization problem. This has reduced in some degree the accuracy of model representation and therefore influenced the performance of the system. A possible way to overcome this problem is to increase the number of models created for faces in one model size. For example, use different models to represent face patterns in different spatial aligning positions. This will, at least improve the detection rate as close to those

pixel domain algorithms as possible.

Second, the classifier we used in this work is still quite simple. Better classifiers such as neural networks and support vector machines are all interesting alternatives to the clustering algorithm used. We plan to try one of them on one model size first to see if continuous works are necessary to apply newer classifier to the multi-model detection framework.

Third, we do not discuss the relation between quantization degradation in DCT parameters and detection accuracy. In our experiments, many testing pictures are downloaded directly from the Internet in compressed JPEG formats. So far, according to our observation, as long as the image quality is acceptable to average human beings, the quantization degradation does not influence detection performance. Though better quantitative results are desirable, a major problem to overcome is how to separate the different factors that influence image quality, *i.e.*, compression loss and problems in the imaging stages.

Chapter 5

Interactive System Design for Video Object Segmentation

5.1. Introduction

In the previous chapters of this thesis, we discussed various model-based algorithms for segmentation and detection of video objects. Model-based solutions have been shown capable of solving the segmentation and detection problems in the constrained application domains and on specific objects such as head-and-shoulders and human faces. However, it is hard to design appropriate models for segmentation purpose in the general sense.

Actually, segmenting and detecting objects from image and/or video data has been under intensive research in the computer vision and image processing community. Numerous algorithms are available in the literature. However, it is becoming widely accepted recently that fully automatic segmentation is difficult. Instead, more and more semi-automatic approaches are designed for the applications in which *fully automation* is not a compulsory requirement. This is especially true for multimedia applications as compared with traditional computer vision applications such as robotic vision, remote sensing, industry automation *etc.* In multimedia industry,

semi-automatic algorithms are especially useful in the media authoring stages. For example, in IBM's HotVideo [109] and HyperVideo from Veon [110] systems, video objects are defined in a semi-automatic way in the authoring stage. Each video object is then associated with a content related hot link that is similar to HTML hotlinks. When the user click on the video objects at the time of playback, the system then jumps to follow the hotlinks. IBM has included in its HotVideo system a browsing tool as well as an authoring tool. Chalom and Bove also designed a semi-automatic segmentation system [12, 11] to identify and track objects at MIT Media Lab, which they later used to create a hyperlink video system [23]. In addition, under MPEG-4 framework, large amount of video object contents have to be created with semi-automatic approaches.

As such, many papers and systems have been published according to the idea of semi-automatic segmentation. For example, [12, 22, 116, 42, 10], *etc.* These papers all sidestepped full automation and designed tracking algorithms to work along with user interaction. However, though their contribution to the "tracking" (algorithm) part of the problem is different, their user interaction models are all very simple, *i.e.*, the user defines a VO in the first frame, and then lets the computer track the VO temporally. This model cannot meet the requirements of interactive authoring tools. For example, a common problem is that no quality criteria are proposed for the computer to detect the loss of tracking and thus ask for additional user input. Instead, the user has to observe the tracking course from time to time and offer new input when he or she finds it necessary.

In this chapter, we propose a new "interpolation" approach for semi-automatic video object segmentation, and discuss an authoring tool prototype design based on this approach. Our focus is to design the algorithm and the user interaction models at the same time, which helps to solve the efficiency problem in interactive

authoring systems. In our system, the user defines a video object by specifying its contour on multiple anchor frames rather than only on the first frame. The computer then uses input information from multiple anchor frames to "interpolate" the VO contours on every frame. Compared with pure tracking approaches, the interpolation approach offers more "interaction points" between the algorithm and the user. From the algorithm's point of view, the algorithm makes use of user input more efficiently, because each user defined contour contributes to VO definitions on those frames before as well as after it, while in the tracking case, a user input only influences the frames after it. From the user's point of view, the new approach makes the system performance more predictable because the user can define a VO on frames where large occlusion or motion occurs and most tracking algorithms are likely to fail. In addition, it is more controllable in that with two or more input VO contours, the possible interpolated contours can be effectively limited to certain searching regions.

More specifically, the problem can be defined as follows. Given two input contours C_b and C_e of a video object on frame b and frame e, try to find the object contours C_i on frames i, $i = b+1, b+2, \cdots, e-1$. We call it a "contour interpolation" problem. As a comparison, the "contour tracking" problem is formulated as: given an input contour C_b, try to find the object contour C_i on frame i, $i = b+1, b+2, \cdots, e$. It is natural to consider an interpolation problem as two tracking problems, *i.e.*, to maximize the use of input information on two frames, we can track the input contour from C_b to C_e also well as from C_e to C_b. In this sense, all the available VO tracking algorithms can be used. However, how to merge the results from these two directional tracking and produce a final best result is obviously an open problem.

To maximize the use of user input on two frames, we use active contour models (snakes) [50] in our interpolation algorithm. In a snake model, a planar curve is

parameterized with nodes and local energy functions are defined for each node. The final shape and position of the curve is determined by the global minimization of the snake's energy. In our work, a traditional snake model is extended in the following ways. First, we use nodes to represent snakes and we design a "contour matching" algorithm to match the node-representations of two user input contours. Based on contour matching, contour temporal smoothness and shape similarity criteria are defined. Later discussion will show that these criteria are essential for merging multiple tracking results. Second, we extend the 2-dimensional snake model to a 3-dimensional model in which new energy terms that reflect spatial temporal constraints are included. Third, a "parametric neighborhood template" is designed to improve the robustness against background edge noises during the active contour tracking course.

The algorithm design in this work is directly influenced by the work [40], in which the idea of contour matching was first proposed. In our work, we further develop their contour matching algorithm by introducing different local motion models. Based on contour matching, the interpolation problem is then modeled as a bi-directional snake tracking problem and a merging problem. In addition, similar work on spatial/temporal active snake model can be found in [60], [36], [16], [59], *etc.* In [60], efforts were made to combine the active contour model with motion estimation. Quality criteria were also suggested to evaluate the quality of contour tracking. However, their experimental results were not good because no node neighborhood information was used. Lin and Kung [59] proposed a concept of *circular viterbi* and applied it to fuse information from motion analysis, edge information and active snakes in order to track and video objects. Chiueh et al. [16] used similar snake technique to track video object contours for annotation. Fu and Tekalp [36] tried to solve the occlusion problem in active contour tracking by segmenting the

contours into multiple segments. They used neighborhood information for motion estimation and got better results than [60]. We will later show in this chapter that their work can be included in ours as a specific case of our parametric template.

The user interface model in this work is related to the work in [81]. In their system, the user selects key points and the computer grows the corresponding contour segment that links the key points. The current key point is moved by the user until the contour segment grown by the computer is desirable. This process involves both the computer's searching as well as the user's decision. It is an efficient model for user/machine interaction. In our system, an improved active searching mechanism (*Interactive Rubberband*) is designed for the user to define the initial VO contours on anchor frames. In addition, an *iterative interpolation* mechanism is designed for the user to offer error feedback to the interpolation algorithm.

This chapter is organized as follows. In Section 5.2., we introduce the contour representation and contour matching algorithm. In Section 5.3., we discuss in detail our contour interpolation algorithm based on contour matching. Then in Section 5.4. we summarize the user interaction models employed in the system. Experimental results are presented in Section 5.5., and in Section 5.6., we present concluding remarks.

5.2. Contour Representation and Matching

5.2.1 Contour Representation

In this work, a contour is represented by a vector array $\{\mathbf{v}_{s,k}\}$, $(s = 0, 1, \cdots, N)$, where k is the temporal location and s is the spatial index of contour pixels. The spatial location of each contour pixel is denoted by vector $\mathbf{v}_s = (x_s, y_s)$. In addition, for concise representation and easy matching, a contour can also be represented with subsampled nodes as $\{\mathbf{v}_{S_k(i),k}\}$, where $S_k : i \mapsto j$, $i \in [0, 1, \cdots, N_s]$, $j \in [0, 1, \cdots, N]$

is the subsampling function related to temporal position k. For example, a uniform subsampling function is defined as $S_k(i) = i \cdot unit$, where $unit$ is the subsampling unit. In this work, we name $\{\mathbf{v}_{s,k}\}$ the *pixel representation* and $\{\mathbf{v}_{S_k(i),k}\}$ the *node representation*. Note that the node representation is actually a first order polynomial approximation of the pixel representation.

5.2.2 Contour Matching

The purpose of contour matching is to find the correspondence between two contours C_b, C_e, which are the contours of the same video object at the different temporal locations (b and e). This is essential for interpolating the object contours between them. In the pixel representation, the length of C_b and C_e is not necessarily the same and pixel by pixel correspondence can not be created. Therefore, we use subsampled node representation for the contour interpolation study and contour matching algorithms are used to find the correspondence between the two subsampled contours.

Mathematically, the matching process can be defined as follows. Given two input contours: $C_b = \{\mathbf{v}_{s,b}\}$, $C_e = \{\mathbf{v}_{s,e}\}$ in the pixel representation, and a subsampled node representation for the first contour $\{\mathbf{v}_{S_b(i),b}\}$, find the corresponding node representation for the second contour $\{\mathbf{v}_{S_e(i),e}\}$. Here the matching of the nodes of two contours can be expressed with the mapping function $f_m : \mathbf{v}_{S_b(i),b} \mapsto \mathbf{v}_{S_e(i),e}$, $i \in [0, 1, \cdots, N_s]$. Generally, we fix the node representation of the first contour C_b by uniform subsampling (Other subsampling algorithms are also possible, see [95] for a detailed discussion), and the matching process is reduced to finding the corresponding subsampling function $S_e(i)$ for the second contour.

In order to find the mapping function f_m, we define a local energy term for each matched node pair. The final matching result is determined by the global minimization of the matching energy of the two contours. This is similar to the

matching approach in [40]. In this work, we assume the motion of the considered video object is nonrigid globally, but rigid locally, and the shape of its two contours is similar locally everywhere. This assumption is true in general if the motion of the video object is not too fast in relation to the temporal distance between the two frames on which the contours C_b and C_e are defined. Two rigid motion models, *i.e.*, translation and affine, are possible for local matching energy definition.

Translation Motion Model: If we denote the motion vector between two matched nodes as $MV_i = \mathbf{v}_{S_b(i),b} - \mathbf{v}_{S_e(i),e}$, then the matching energy term is defined as

$$E_i = \mu||MV_i - MV_{i-1}|| + \eta(S_e(i) - S_e(i-1))^2$$

where the first term is the smoothness evaluation of the motion vectors of two neighboring nodes, the second term is an elastic constraint on the distance of two neighboring nodes, and μ, η are two weighting factors.

Affine Motion Model: In contrast to the translation model, the affine model needs three motion vectors to define. Here we denote the affine mapping function as $\mathcal{A}_i = \text{Affine}_i(MV_{i-1}, MV_i, MV_{i+1})$. Under this function, the local contour segment $\{\mathbf{v}_{s,b}\}$, $s \in [S_b(i-1), S_b(i+1))$ is projected to a pixel set $\{\mathbf{v}'_e(s)\}$, while its corresponding contour segment in frame e is denoted as pixel set $\{\mathbf{v}_{S_e(s),e}\}$, $s \in [S_e(i-1), S_e(i+1))$. Then the matching energy term is defined as

$$E_i = \mu D\Big(\{\mathbf{v}'_e(s)\}, \{\mathbf{v}_{S_e(s),e}\}\Big) + \eta(S_e(i) - S_e(i-1))^2,$$

where the operator $D(\cdot)$ is the distance measure of two sets, which we define as

$$D(A,B) = \Big[\sum_{\mathbf{v}_i \in B} \min_{\mathbf{v}_j \in A} ||\mathbf{v}_i - \mathbf{v}_j|| + \sum_{\mathbf{v}_j \in A} \min_{\mathbf{v}_i \in B} ||\mathbf{v}_j - \mathbf{v}_i|| \Big] / 2. \tag{5.1}$$

In practice, Eq. (5.1) can be implemented in an iterative manner for each pixel. Here we define $d(\mathbf{v}_i, A) = \min_{\mathbf{v}_j \in A} ||\mathbf{v}_i - \mathbf{v}_j||$, and denote $d_n(\mathbf{v}_i, A)$ as the n-th value for $d(\mathbf{v}_i, A)$ in the iteration. At the initialization stage,

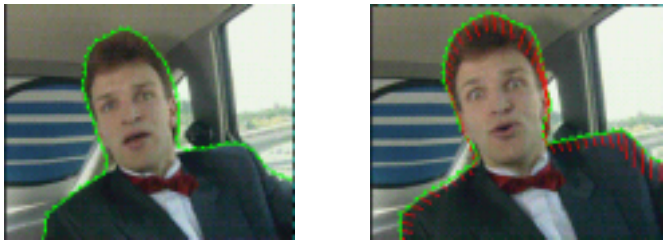
$$d_0(\mathbf{v}_i, A) = \begin{cases} 0 & \text{if } \mathbf{v}_i \in A \\ +\infty & \text{otherwise} \end{cases}.$$

The iteration is then defined as

$$d_{n+1}(\mathbf{v}_i, A) = \min_{\mathbf{v}_j \in N(\mathbf{v}_i)} \Big(d_n(\mathbf{v}_j, A) + ||\mathbf{v}_i - \mathbf{v}_j|| \Big),$$

where $N(\mathbf{v}_i)$ is the 8-neighborhood set of pixel $\mathbf{v}_i$.

With the definition of local matching energy terms, the matching problem is converted to an energy minimization problem. This can be easily solved by the DP algorithm [7], which we do not discuss in detail here. In practice, the affine model is more complex than the translation model, but the quality is better, especially when the subsampling distance between neighboring nodes is big. In Fig. 5-1, we show a result of contour matching under the translation model. Fig. 5-1(a) is the 50th, 5-1(b) is the 70th frame of the Carphone sequence. The green lines are the contours and the red lines link the matched node pairs.



(a) (b)

Figure 5-1: Results of contour matching for the Carphone sequence. (a) is the 50th and (b) is the 70th frame of the Carphone sequence. The green lines are the user specified contours and the red lines link the matched node pairs.

5.3. Contour Interpolation Algorithm

5.3.1 Localized Energy Minimization Model

The essence of Kass' snake model [50] is to define a local energy term for each contour node, and the shape of the contour is determined by minimizing the total snake energy globally. When we go from a 2-dimensional spatial snake to 3-dimensional spatial/temporal snake, it is possible to extend the local energy terms from 2D to 3D as well. In this work, we study the extended local energy terms as intraframe energy terms and interframe energy terms, respectively. From now on, because no subsampling issue will be involved, snakes are assumed to be in the node representation. The node representation notation $\{\mathbf{v}_{S_k(i),k}\}$ is simplified to $\{\mathbf{v}_{i,k}\}$ whenever possible.

Intraframe Energy: As usual, the intraframe energy terms include two parts, a gradient term and a smoothness term:

$$E_{\text{intra},i} = \eta_{\text{edge}} E_{\text{gradient},i} + \eta_{\text{smooth}} E_{\text{smooth},i}. \tag{5.2}$$

In this equation, the first gradient term is defined as

$$E_{\text{gradient},i} = \int_{s=S(i-1)}^{S(i)} \frac{255}{(10 + || \bigtriangledown (\mathbf{c}(\mathbf{v}_s))||)} ds,$$

in which 255 and 10 are two empirical values. $\mathbf{c}(\mathbf{v})$ is the color vector of node $\mathbf{v}$. The second smoothness term is defined as

$$E_{\text{smooth},i} = \Big(\frac{2||\mathbf{v}_{S(i-1)} + \mathbf{v}_{S(i+1)} - 2\mathbf{v}_{S(i)}||}{||\mathbf{v}_{S(i-1)} - \mathbf{v}_{S(i+1)}|| + ||\mathbf{v}_{S(i)} + \mathbf{v}_{S(i+1)}||} \Big)^2,$$

which is designed to eliminate the influence of the node distance on the contour smoothness measure.

Interframe Energy: In available papers on temporal active contour tracking [60, 36], generally employed interframe energy terms include optical flow, motion smoothness, interframe color, *etc.* A basic problem in these approaches is that most of their node energy definitions are based only on image features at the node's position rather than on its neighborhood. This makes the color and especially motion information generally not accurate. Ideally, the contour nodes' neighborhood should be observed in order to track them from frame to frame. However, a problem here is, different from typical point tracking problems, contour nodes are most likely located on the boundary of a moving object, so their neighborhood is not constant. In order to capture the consistency through the tracking course, we introduce a concept of *Parametric Neighborhood Template*. A parametric template T is defined as a data structure including two arrays: a vector array $\{\mathbf{dv}_0, \mathbf{dv}_1, \cdots, \mathbf{dv}_n\}$ and a weighting array $\{w_0, w_1, \cdots, w_n\}$, in which a vector $\mathbf{dv}_I$ represents the position of the i-th pixel in the neighborhood and w_i represents its contribution weight to template measurements. For each contour node $\mathbf{v}_{i,k}$, a parametric template $T_{i,k}$ is

defined and kept updated frame by frame through the tracking course.

With the definition of parametric template $T_{i,k}$, the color of two temporally neighboring contour nodes $\mathbf{v}_{i,k}$, $\mathbf{v}_{i,k-1}$ can be compared as

$$\begin{aligned}&\text{Diff}_c(\mathbf{v}_{i,k}, \mathbf{v}_{i,k-1}, T_{i,k}) = \\ &\qquad \sum_{\mathbf{dv}_j \in T_{i,k}} w_j ||\mathbf{c}(\mathbf{v}_{i,k} + \mathbf{dv}_j) - \mathbf{c}(\mathbf{v}_{i,k-1} + \mathbf{dv}_j)||, \end{aligned} \tag{5.3}$$

where $\mathbf{c}(\mathbf{v})$ is the color vector of node $\mathbf{v}$. In addition, we define the motion vector at node $\mathbf{v}_{s,k}$ as

$$MV(\mathbf{v}_{i,k}, T_{i,k}) = \sum_{\mathbf{dv}_j \in T_{i,k}} w_j \cdot MV^{(p)}(\mathbf{v}_{i,k} + \mathbf{dv}_j) / \sum_{\mathbf{dv}_j \in T_{i,k}} w_j, \tag{5.4}$$

where $MV^{(p)}(\mathbf{v}_{i,k})$ is the estimated motion vector at $\mathbf{v}_{i,k}$. If we do not consider occlusion, a simple way to determine the weights of template T is to set w_j for those neighboring pixels inside the contour as 1 and for those outside the contour as 0. This can be illustrated in Fig. 5-2. For the occlusion case as was discussed in [36], we can easily switch the weights by setting inside weights to 0 and outside weights to 1. In general cases, both the size and the weights of the parametric template can be adjusted flexibly to get the best results of the contour node tracking.

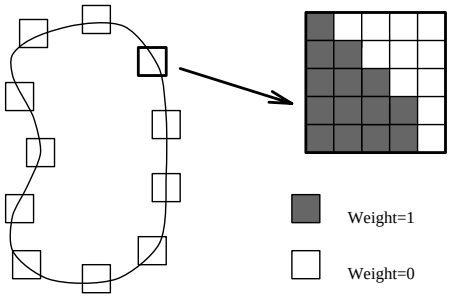


Figure 5-2: Illustration of parametric template concept. During the tracking course, both the size and weights of the template can be adapted.

Based on parametric template, the interframe energy terms for each contour node are enumerated as follows.

1. Color Similarity: $E_{\text{color},i,k} = \text{Diff}_c(\mathbf{v}_{i,k}, \mathbf{v}_{i-1,k}, T_{i,k})$. Here function $\text{Diff}_c()$ is defined in Eq. (5.3).

2. Optical Flow: $E_{\text{optical},i,k} = MV(\mathbf{v}_{i,k-1}, T_{i,k-1}) - MV_{i,k-1}$. Here the first term $MV(\mathbf{v}_{i,k-1}, T_{i,k-1})$ is defined in Eq. (5.4) and we assume forward motion estimation is used. The second term is $MV_{i,k-1} = (\mathbf{v}_{i,k} - \mathbf{v}_{i,k-1})$.

3. Motion Smoothness: $E_{\text{motion},i,k} = ||MV_{i-1,k} + MV_{i+1,k} - 2MV_{i,k}||$. This is a smoothness measure for motion vectors on the spatially neighboring nodes. It is accurate for local motion in the translation form.

4. Shape Stiffness: $E_{\text{shape},i,k} = ||\text{angle}(\mathbf{v}_{i-1,k-1}, \mathbf{v}_{i,k-1}, \mathbf{v}_{i+1,k-1}) - \text{angle}(\mathbf{v}_{i-1,k}, \mathbf{v}_{i,k}, \mathbf{v}_{i+1,k})||$, where $\text{angle}(\mathbf{v}_{i-1}, \mathbf{v}_i, \mathbf{v}_{i+1})$ is the angle based on the three spatially neighboring nodes. This term measures the local shape similarity. It is accurate if the local motion is in the rotation form.

5. Temporal Smoothness: $E_{\text{temporal},i,k} = ||\mathbf{v}_{i,k-1} + \mathbf{v}_{i,k+1} - 2\mathbf{v}_{i,k}||$. This is the smoothness measure for the three temporally neighboring nodes.

The interframe energy E_{inter} is then the weighted sum of above terms.

Search Algorithm for Minimization: With the definition of local energy terms, the contour interpolation problem can be expressed as an energy minimization problem as follows. Given two contours $\{\mathbf{v}_{i,b}\}, \{\mathbf{v}_{i,e}\}$, $(b < e, i = 0, 1, \cdots, N_s)$, find the contour nodes $\{\mathbf{v}_{s,k}\}$, $(k = b+1, \cdots, e-1, s = 0, 1, \cdots, N_s)$, that minimize the global energy $\sum_{s,k}(E_{\text{inter},s,k} + E_{\text{intra},s,k})$.

Though it is natural to extend the local energy terms from 2D to 3D, the increase in computational complexity is an important problem. In the 2D case, each node $\mathbf{v}_i$'s

local energy is defined in relation to two neighbors, *i.e.*, $E_{\text{intra}} = f(\mathbf{v}_{i-1}, \mathbf{v}_i, \mathbf{v}_{i+1})$, while in the 3D case, the local energy terms for each node $\mathbf{v}_{s,k}$ is defined in relation to eight neighbors! This change is illustrated in Fig. 5-3. If each node has a search region of n, the local searching complexity then increases from n^3 to n^9, which makes global minimization algorithm difficult to design. Though powerful algorithms such as *simulated annealing* should still be able to solve this minimization problem, the slow speed of convergence makes it inappropriate for an interactive segmentation tool.

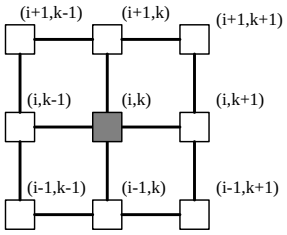


Figure 5-3: Spatial temporal neighborhood of a contour node for the local energy definition. In the figure, i is the spatial index and k is the temporal index.

In this work, a sub-optimal solution is obtained by converting the contour interpolation problem into two tracking problems: a forward tracking from C_b to C_e and a backward tracking from C_e to C_b. They are also referred to as bi-directional tracking in this work. When converted to a tracking problem, the neighborhood definition of the current pixel (i, k) cannot include nodes in a neighboring frame that has not been processed. Therefore, we shift the neighborhood definition in Fig. 5-3 by one frame. For example in the forward tracking model, if the current node is (i, k) in Fig. 5-3, then its eight nodes are $(i+1, p), (i, p), (i-1, p), p = k-2, k-1, k$ (not including (i, k)). Among them, six are already fixed and only two variable nodes $(i+1, k+1)$ and $(i-1, k+1)$ will influence its local energy. This is the same as the case with the intraframe energy $E_{\text{intra},s}$. Therefore, it is easy to design a

search algorithm with DP. After the bi-directional tracking, another search process is used to find the optimal contours out of the previous tracking results.

5.3.2 Bi-directional Tracking

In the tracking model, each node $\mathbf{v}_{i,k}$'s total node energy can be written as

$$E_{\text{total},i,k} = f_E(\mathbf{v}_{i-1,k}, \mathbf{v}_{i,k}, \mathbf{v}_{i+1,k}) \tag{5.5}$$

because the other six neighboring nodes $\mathbf{v}_{i+p,k-2}$, $\mathbf{v}_{i+p,k-1}$, $p = (-1, 0, 1)$ in the previous frames are all fixed. This energy expression is similar to those used for 2D active contour models, except that the detailed expression of f_E is different. Therefore, we use the DP algorithm similar to that used in [7].

Limited Search Region v.s. Searching Complexity According to [7], the local energy expression in Eq. (5.5) is a second order expression in the sense that it includes two neighboring nodes as variables. In this case, a two-element vector $(\mathbf{v}_{i+1}, \mathbf{v}_i)$ is used as the status index for DP. The DP search is then carried out as follows

$$S_i(\mathbf{v}_{i+1}, \mathbf{v}_i) = \min_{\mathbf{v}_{i-1}} \Big[S_{i-1}(\mathbf{v}_i, \mathbf{v}_{i-1}) + f_E(\mathbf{v}_{i-1}, \mathbf{v}_i, \mathbf{v}_{i+1}) \Big].$$

Note that the frame index k is omitted in the above expression because no temporal information is involved. If the search size for each node is n and the total number of nodes in each contour is m, the complexity of DP is $O(mn^3)$. Obviously, the search size n is an important factor in the overall complexity and should be limited as much as possible.

Two clues are used to limit the search size in our work. First at the global level, a search stripe can be created for each matched node pair. This is illustrated in Fig. 5-4. For ease of discussion, the temporal orbit of the matched node is projected

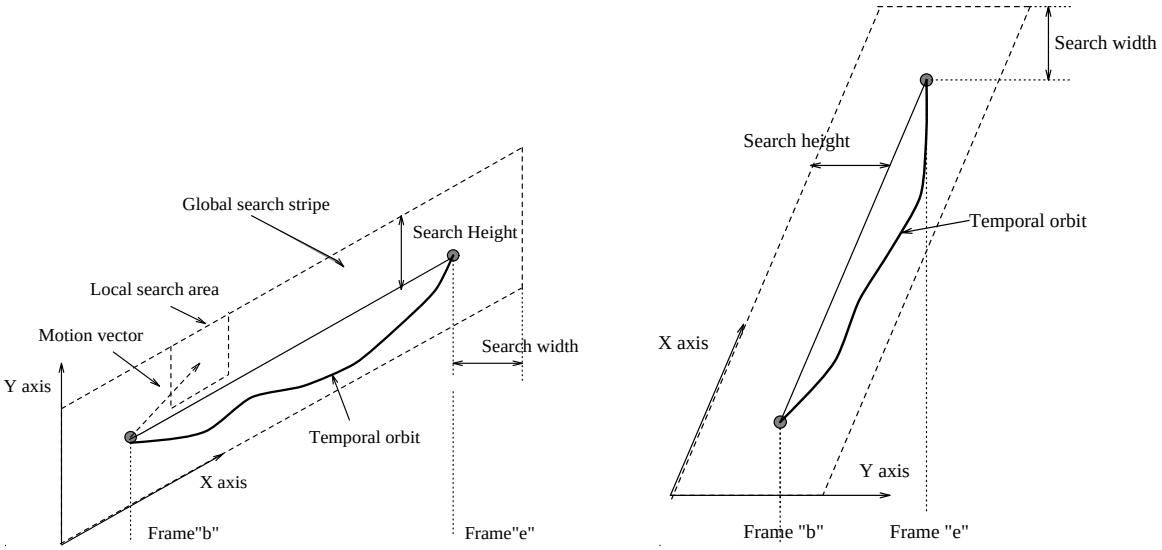


Figure 5-4: Global and local search region limitation. The two grayed circles represent the matched node pair. The size of the global search region is determined by "search height" and "search width", while the local search region is determined by motion estimation.

into one frame. On this frame, a search stripe is defined. Note that the definition of a search stripe is different according to different spatial location of the matched node pairs. In Fig. 5-4(a), and 5-4(b), two basic types of search stripe definitions are depicted. The orientation of width and height are differently defined as well based on the different orientations of the matched node pairs. In practice, both the width and height of the global search stripe can be determined by the user in an interactive way, according to the motion of video object. That is, if the motion is more like a pure translation, the height of the stripe $S^{(G)}_{\text{height}}$ may be reduced, otherwise it is increased. The bottom line is that the global search region should be at least big enough to cover the temporal orbit of the node's motion. In addition, in the case of large search stripes that exceed a certain threshold, the global search stripe is further re-sampled to reduce the overall search size. The two axes "x" and "y" used for resampling are marked in Fig. 5-4(a) and 5-4(b). Note that the

parallelogram search region definition is more efficient than a strictly rectangular one in the computational sense, while without much performance degradation.

Once the global stripes are defined for node pairs, the tracking of nodes is constrained within their stripes on every frame. In addition, at the local level, the local search region is further determined frame by frame by the forward motion vector at the current frame. In our work, the determination of $S^{(L)}_{\text{height}}$ and $S^{(L)}_{\text{width}}$ is based on $S^{(G)}_{\text{height}}$ and $S^{(G)}_{\text{width}}$, and the local search is carried out on top of the re-sampled grids created for the global search stripe, *i.e.* the re-sampled grids along the "x" and "y" axes.

In above two clues, the global stripe limits the possible location of the local search region. Note that in the pure "interpolation" case, both the width and height of the global search stripe is zero. Therefore, the global search stripe is actually a generalization of interpolation.

Closed Contour Problem Until now, the discussed contours C_i are all by default open. In practice when contours are constrained to be closed, the minimization search used for tracking purpose can be approximated with two-pass open contour searching. That is, for a closed contour $C_i, i = 0, 1, 2, \cdots, n$, node $C_0 = C_n$, its minimization status can be found as follows. First break the closed contour at node C_0, use previous algorithm to process open contour $C_i, i = 0, 1, 2, \cdots, n-1$, which produces a temporary contour C'_i. Second, close C'_i by setting $C'_n = C'_0$ and then break it half way at node $C'_{(n/2)}$, this produces an open contour C''_i. C''_i is further processed with the open contour algorithm and the final result is obtained.

Figure 5-5: Illustration of the necessity of bi-directional tracking and result merging. (a), (b) are the user defined VO on the 118th and the 132nd frame of the Carphone sequence. (c) is the tracked object contour on the 125th frame by forward tracking. (d) is the tracked object contour on the 125th frame by backward tracking.

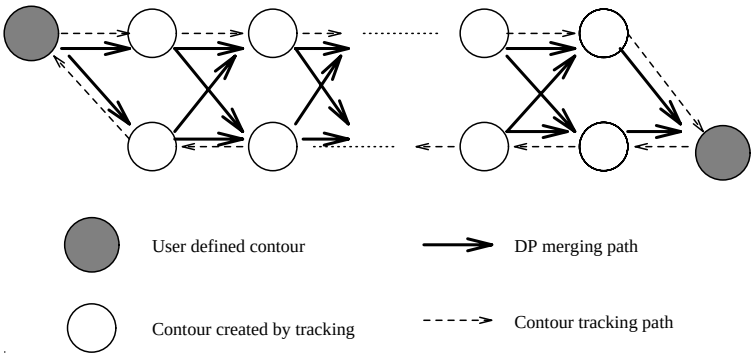


Figure 5-6: Illustration of DP approach for merging the bi-directional contour tracking. Each circle represents a contour $C_k^{(d)}$, DP searching is used to find an optimal path in the temporal direction that has the best merit quality.

5.3.3 Merging of Multiple Results

Though the bi-directional tracking approach reduces the searching complexity, its limitation is that it only makes use of user input in one frame (either C_b or C_e) at a time. Due to the error accumulation, the tracked contour C_k always degrades when k approaches e, if tracked from C_b to C_e, and vice versa. Actually we have observed that if the object's motion involves self-occlusion and/or uncovering, sometimes it is very difficult for the active contour model to track its contour in one direction, but easy to do it in the other direction. Fig. 5-5 is such an example. In Fig. 5-5, (a), (b) are the user defined VO on the 118th and the 132nd frame of the Carphone sequence. From frame 118 to frame 132, the man's left ear was uncovered because of the rotation of his head. If we track the object contour forward, *i.e.* from the 118th frame to the 132nd frame, the uncovered ear was not included as part of the video object. This is depicted in Fig. 5-5(c). On the other hand, if we track the contour backward, *i.e.* from the 132nd frame to 118th frame, the motion is reversed and the uncovering motion of left ear changes to occlusion, which is easy for the active contour tracking algorithm to handle. The result of backward tracking for frame 125 is shown in Fig. 5-5(d). Note that the left ear is correctly included as part of the video object.

Therefore, a good algorithm to merge the results from the two tracking processes is important. In this work, an efficient DP algorithm is designed for the merging job. The problem may be defined as follows. Given two set of contours $\{C_k^{(1)}\}$, $\{C_k^{(2)}\}$, $(k = b, b+1, \cdots, e)$, that are created by two contour tracking processes (one from C_b to C_e and the other from C_e to C_b), find a contour set C_k, $(k = b, b+1, \cdots, e,\ C_k = C_k^{(1)}$ or $C_k = C_k^{(2)})$ that meets certain *merit criteria* as the final output of contour interpolation. In the terminology of DP, we can say that the target is to find an optimal path from C_b to C_e that maximize the merit criteria.

In this work, the *merit criteria* for each candidate contour include two terms: a temporal smoothness energy term E_T, and a shape merit energy term E_S, *i.e.*

$$E_{\text{merge}}(C_k) = \eta_T \cdot E_T(C_k) + \eta_S \cdot E_S(C_k). \tag{5.6}$$

The first term is defined in a localized form:

$$E_T(C_k) = \sum_{i=0}^{i=N_s} ||\mathbf{v}_{i,k-1} + \mathbf{v}_{i,k+1} - 2\mathbf{v}_{i,k}||, \tag{5.7}$$

where N_s is the number of nodes in each contour. Note that $E_T(C_k)$ is different from previously defined $E_{temporal}$ in that $E_T(C_k)$ is defined for each contour while $E_{temporal}$ is defined for each contour node.

The second term of Eq. (5.6) is based on the shape similarity comparison between two contour pairs: (C_b, C_k) and (C_k, C_e). If we denote the shape similarity measure of two contours C_k and C_l as $\text{shape}(C_k, C_l)$, then the E_S term can be written as

$$E_S(C_k) = \Big[w_1(k) \cdot \text{shape}(C_b, C_k) + w_2(k) \cdot \text{shape}(C_e, C_k)\Big], \tag{5.8}$$

where $w_1(\cdot)$ and $w_2(\cdot)$ are two weighting functions. They are designed as a linear function of frame indices k, b, and e. In addition, the shape similarity measure used here can be defined based on two interframe node energy terms E_{shape} and E_{motion}, as discussed in Section 5.3.1. At the contour level, this expression is defined as

$$\text{shape}(C_k, C_l) = \sum_{i=0}^{i=N_s} \eta_1[(\mathbf{v}_{i,k} - \mathbf{v}_{i,l}) - (\mathbf{v}_{i-1,k} - \mathbf{v}_{i-1,l})]$$

$$+\eta_2[\text{angle}(\mathbf{v}_{i-1,k}, \mathbf{v}_{i,k}, \mathbf{v}_{i+1,k}) - \text{angle}(\mathbf{v}_{i-1,l}, \mathbf{v}_{i,l}, \mathbf{v}_{i+1,l})].$$

In the above Eq.s (5.6)-(5.8), $E_{merge}(C_k)$ is actually defined in relation to three

contours: C_{k-1}, C_k and C_{k+1}. Therefore, it is easy to solve the minimization problem with DP. Here Fig. 5-6 is used to illustrate the DP based merging algorithm. In Fig. 5-6, each circle represents a contour $C_k^{(d)}$, DP searching is used to find an optimal path in the temporal direction that has the best merit quality, *i.e.*, in the sense of temporal smoothness and shape similarity.

5.4. User Interaction Model

In a typical semi-automatic system, the user's role includes two important functions. One is to give the initial data for the computer to begin the computation, the other is to correct the computer's errors. In our system, these two functions are handled by *Interactive Rubberband* and *Iterative Interpolation*, respectively.

5.4.1 Interactive Rubberband

In available image authoring systems, two types of solutions are common for a user to specify an object contour in an image (or a video frame). In one of them (Type One), the user specifies every point of the contour, while the computer does nothing but record the positions of each mouse click and links the positions with line segments. Typical examples include the polyline drawing in XFIG and free style drawing in PHOTOSHOP. This type of solution gives the user full control of the shape and position of the contour, but ignores the computational power of the computer. Obviously, it is tedious to input an accurate contour point by point. On the other hand, in the other type of solutions (Type Two), the image is modeled as grids and the contour as paths linking the grids. The user has only to select a starting point and an ending point of a contour segment, the computer finds the whole segment by searching the minimal cost path that links the two points. This can be done with either dynamic programming or graph searching algorithms such

as Dijkstra's algorithm [21]. Publications belonging to this type include [81], [72], [80], *etc.* Compared with Type One solutions, Type Two approach relieves the user's labor by introducing computer searching during the interaction. However, its problem is that the user has less control on the contour. Sometime when the gradient information within the image is complex, an intended contour segment may be attracted to an erroneous strong neighboring edge, which is totally undesirable. In addition, the "Active-Scissors" approach defined in [81] requires the computer to calculate the optimal path to every pixel within the image every time a new contour point position is chosen by the user. This is not efficient if the size of image is big and real time performance is hard to achieve.

To overcome the problems while retaining the benefits of above two groups of solutions, we design an *Interactive Rubberband* tool, which is a hybrid of the two of them.

An Interactive Rubberband is in principle a dynamic graph searching edge detection algorithm that is similar to the second of the above mentioned two types of solutions. The difference is that it comes with an adjustable containing rectangle that limits the range of graph searching. This is illustrated in Fig. 5-7. The user moves the current moving point, which, together with the fixed point, determines a containing rectangle. Graph searching is then carried out within the rectangle and a contour segment is grown to link these two points. Compared with the Active-Scissors approach in [81], Interactive Rubberband is more efficient because it limits the graph searching range to the neighborhood of the desirable contour segment. Moreover, the user can control the result of graph searching by adjusting the width of the rectangle, *e.g.*, if there is strong noisy edges in the neighborhood, the user may get rid of them by narrowing the containing rectangle. In the extreme case, the width can be set to one, then the Interactive Rubberband reduces to above type

one solution. In this sense, the Interactive Rubberband is a generalization of the previous type one and type two solutions.

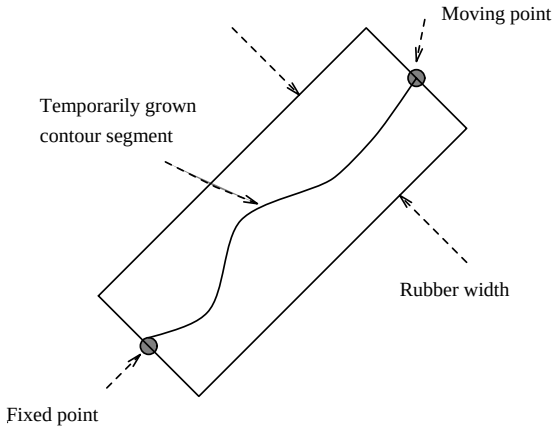


Figure 5-7: Illustration of active rubberband. The containing rectangle is determined by two points: a fixed point and a moving point, and the rubberband width. The width of the rubberband is adjustable by the user.

5.4.2 Iterative Interpolation

Though in experiments the discussed interpolation algorithm showed good performance, error is inevitable in practice, especially when the two anchor frames on which the user specifies object contours C_b and C_e are far away in the temporal direction. Iterative interpolation is found to be a good solution to this problem.

In general, the reasons for errors in interpolation are complex. However, in the interpolation algorithm based on bi-directional tracking, a video object contour can always be reliably tracked from C_b to a certain $C_{(b+t_1)}$, and from C_e to a C_{e-t_2}, where $t_1 > 0$ and $t_2 > 0$. If $b+t_1$ turns out equal to $e-t_2$, the problem is solved. Otherwise, we can move the C_b to $C_{(b+t_1)}$, and C_e to C_{e-t_2}, and begin the interpolation again. In this way, the interpolation is done iteratively until the two contours C_b and C_e converge. This process can be better illustrated in Fig. 5-8. In Fig. 5-8, points

$P(1)$ and $P(2)$ are a matched node pair on contours C_b and C_e. After the first round of interpolation, point $P(1)$ is correctly tracked to $P(3)$ and $P(2)$ to $P(4)$. At this stage, the user changes the C_b to the temporal position at $P(3)$ and C_e to the position at $P(4)$, sets the global search parameters accordingly, and begins the interpolation again. As depicted in the figure, the global search area in the second round for point pair $P(3) - P(4)$ is much smaller than that of $P(1) - P(2)$. This is an important factor that helps the iterative interpolation process converge.

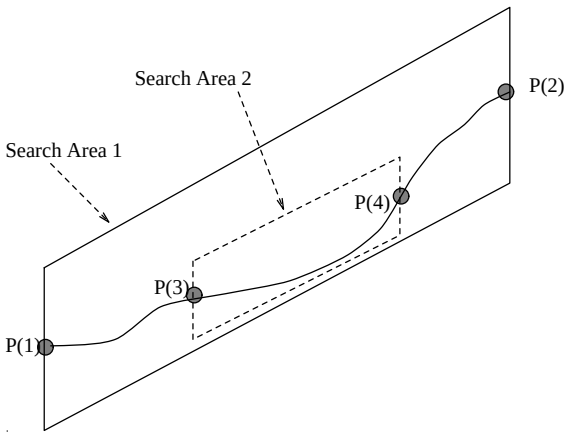


Figure 5-8: Illustration of iterative interpolation. Point $P(1)$ and $P(2)$ are a matched node pair on initial contours C_b and C_e. After the first round of interpolation, point $P(1)$ is correctly tracked to $P(3)$ and $P(2)$ to $P(4)$. Their corresponding contours are used as new C_b and C_e and the interpolation is done iteratively until converges.

In Fig. 5-9, a practical example is given on the Foreman sequence to show how the iterative interpolation works. In the first interpolation round, the user specifies C_b on frame 50 and C_e on frame 100, the global searching parameters are $height = 15$(pixels), $width = 4$(pixels). Subfigures (a) to (e) are results on frames 80, 84, 87, 90, 94 in this round. Obviously, the tracking result is poor from frame 80 to 94, mainly because a strong neighboring edge has attracted the snake erroneously (due to space limits, other frames are not included in the figure). To overcome the error, the user moves C_b to frame 80 and C_e to frame 94, and change the global

searching parameters to $height = 4$, $width = 4$. Based on the contours on frame 80 and 94, the results after the new interpolation round on frames 84, 87, 90 are shown in subfigures (f), (g), (h), respectively. Obviously, the second interpolation round improves the accuracy of tracked contours if we compare the contours in subfigures (b), (c), (d) with those in (f), (g), (h). It is worth noticing that in the second round of interpolation, the user does not have to tell the computer laboriously what exactly a correct contour is. Instead, the user just chooses new anchor frames on which the tracked contours are correct, and changes the global searching parameters accordingly (limits the global searching area as much as possible). It is the computer's work to do the interpolation again based on new user input information!

5.5. Experiments

The discussed video object annotation system is implemented on PCs running Windows 95. Experiments are carried out over MPEG-4 testing video sequences as well as sequences from a library used by Columbia's VideoQ[1] system.

First, we compare the effect of parametric template for active contour tracking on Carphone sequence. In the experiment, single direction forward tracking is used. In Fig. 5-10, the left column is the tracking result without and the right column is the result with the parametric template. The first row is the beginning user input contour in frame 0, the following two rows are the tracked results for the 43rd and 47th frames. Obviously, the parametric template improves the tracking robustness in complex boundary conditions.

Fig. 5-11 shows the results of tracking result merging on the Mother-Daughter sequence. The user defines VO contours on frame 75 and 180. The first row is the result of backward tracking from frame 180 to frame 75, and the second row is the

[1] `http://www.ctr.columbia.edu/VideoQ`

interpolation results that merge bi-directional trackings. The frame numbers, from left to right, are 170, 160, 150, 130 and 90. We can see that from frame 180 to 160, the merged results are taken from backward tracking, while from 160 to 75, the merged results are taken from forward tracking (which is not shown here due to the space limit). That is, the merged interpolation results are better than tracking results on either direction.

Fig. 5-12 is a fully finished segmentation result on the first 100 frames of the Foreman sequence. Subfigures (a)-(h) are the produced contours on frame 10, 20, 30, 40, 60, 70, 80, 90. To finish the segmentation, the user specified three initial contours on frame 0, 50 and 100. One additional iteration is involved in the interpolation between frame 0 and frame 50, and frame 50 and frame 100, respectively.

In practice, the computational complexity of the algorithm depends on the size of the searching area and the complexity of the contours. In our experiment, a 200-MHz Pentium is used, the average speed of the interpolation algorithm is about 0.01 second/node and/or 0.6 second/frame (tested on several MPEG-4 sequences in QCIF size).

5.6. Concluding Remarks

In this chapter, an interactive authoring system is designed for video object segmentation and annotation. This system features a new contour interpolation algorithm, which makes better use of user inputs and has more stable performance than traditional single directional tracking algorithms. In addition, efficient user interaction models are built for both initial data input and machine error feedback. Our experiments show that this prototype system works efficiently both from the machine's and the user's point of view, in that it elegantly balances the use's decision making capability and the computer's processing and searching power.

(a)

(e)

(b)

(f)

(c)

(g)

(d)

(h)

Figure 5-9: An example of iterative interpolation on the Foreman sequence. In the first round, the user specifies C_b at frame 50 and C_e at frame 100, the global search parameters are $height = 15$(pixels), $width = 4$(pixels). (a) to (e) are results on frames 80, 84, 87, 90, 94 at this round. In the second round, previous contour result on frame 80 is used as C_b and contour result on frame 94 is used as C_e, the global search parameters are reduced to be $height = 4$, $width = 4$. The new interpolation results on frame 84, 87, 90 are showed in (f), (g), (h), respectively.

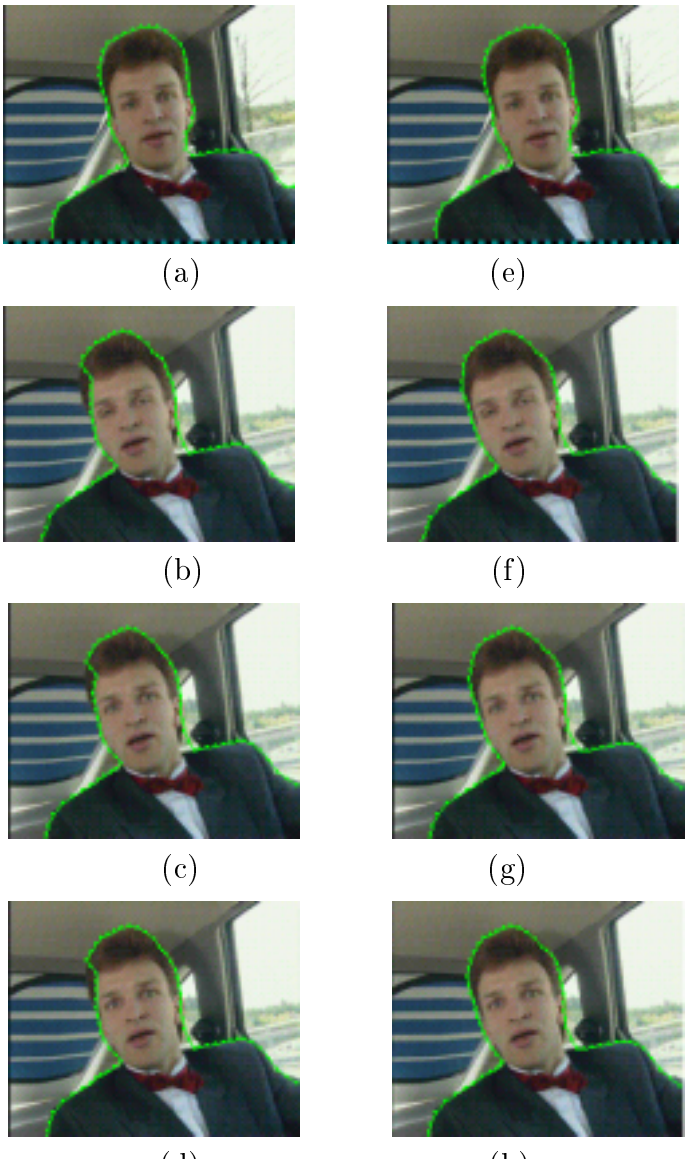


Figure 5-10: Comparison of the effect of parametric template on active contour tracking. The left column is the tracking result without and the right column is the result with parametric template. First row is the beginning user input contour in frame 0, the following two rows are the tracked results for the 43rd and 47th frames.

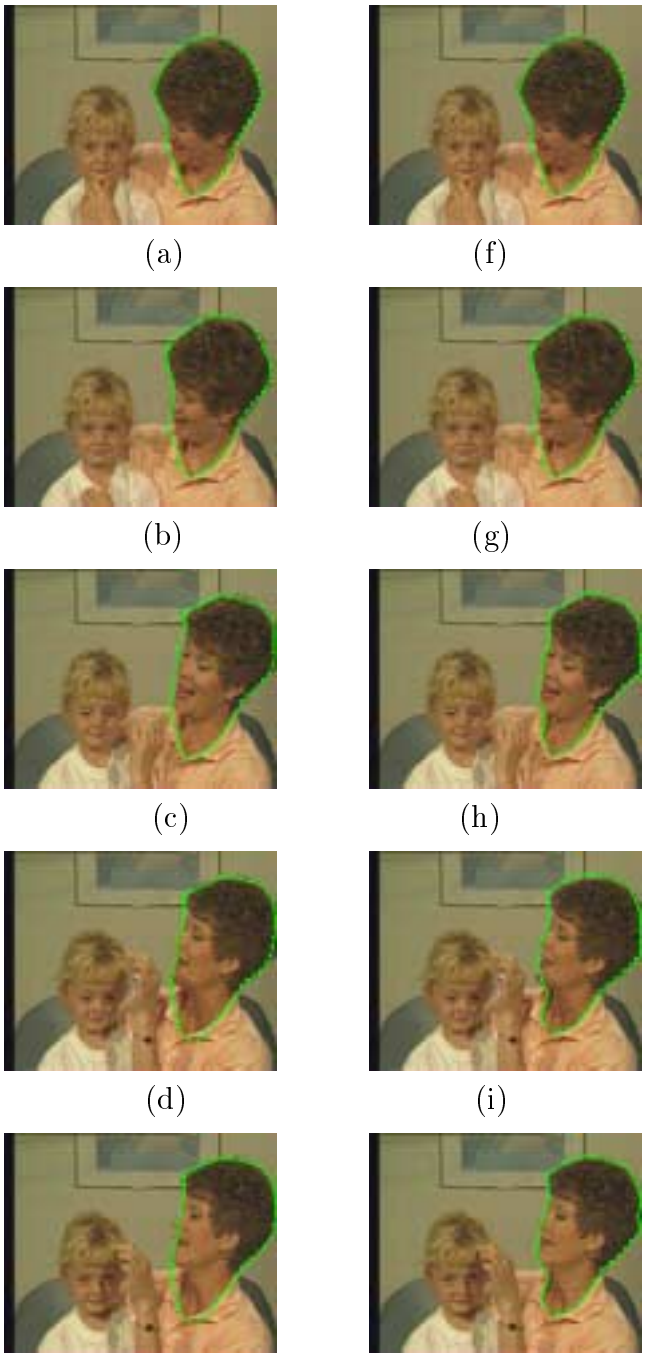


Figure 5-11: Illustration of tracking results merging. The user defined VO contours on frame 75 and 180. The first column is the result of backward tracking from frame 180 to frame 75, and the second column is the interpolation results that merged bi-directional tracking results. The frame numbers, from top to bottom, are 170, 160, 150, 130 and 90.

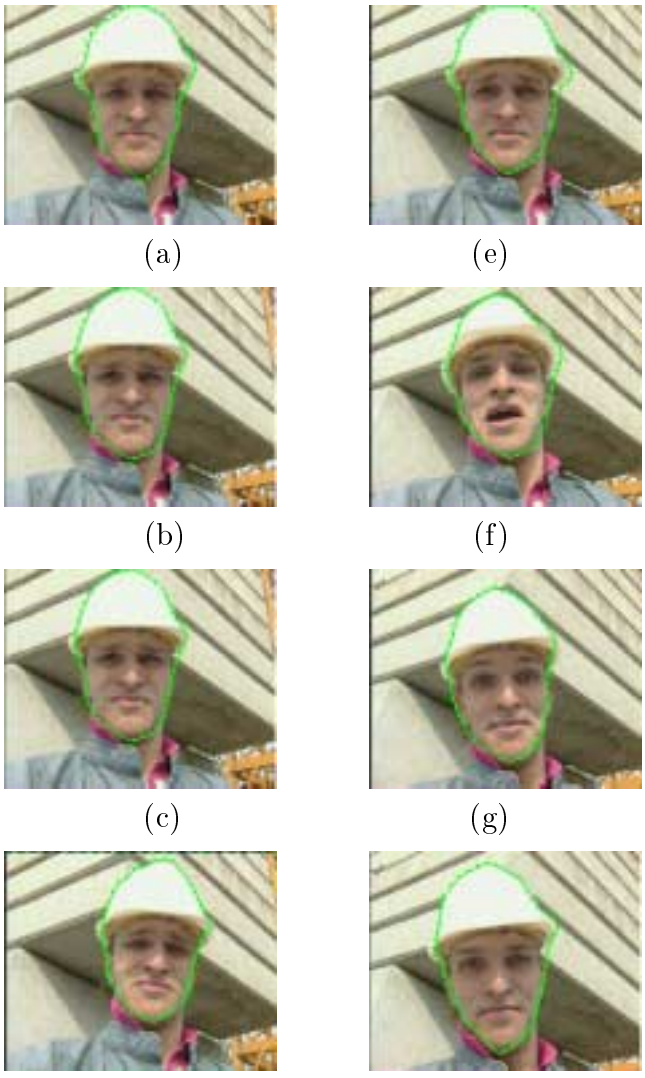


Figure 5-12: Illustration of a full segmentation result on the Foreman sequence. The frame numbers are 10, 20, 30, 40, 60, 70, 80, 90 for subfigures (a)-(h).

Chapter 6

Conclusion and Future Work

In this thesis, we have designed several analysis algorithms for video object detection and segmentation in the general framework of multimedia content analysis. The algorithms are developed in two directions, *i.e.*, fully automatic algorithms and semi-automatic algorithms.

In automatic analysis research, our works are mainly focused on model-based algorithm design. Because video content analysis has been a difficult problem in image understanding and computer vision research, general-purpose algorithms are hard to design. Instead, in our work, we have tried to design detection and segmentation algorithms based on different multimedia applications. By constraining the application domain or the problems to be solved, we could represent the video objects to be detected or segmented with some well-defined templates and/or models. This way, the difficult video analysis problems become tangible with reasonable computation resources. Specifically, we have designed video object detection and/or segmentation algorithms for three multimedia applications: (1) Real-time VO segmentation for videophones and videoconferences. (2) Automatic anchorperson detection and segmentation for broadcasting news indexing and retrieval. (3) Automatic face detection from the compressed DCT domain. All of them are highly structured patterns and have strong pattern features with respect to color, texture

and shapes.

For videoconference applications, we have designed a foreground and a background model and an online tracking mechanism to create and track the model parameters for both foreground and background models. In our system, the background model is a simple 2D array of pixels, each has an independent Gaussian distribution. The head-and-shoulder pattern foreground is modeled with a blob-based region model and a shape model. Both models have a Gaussian assumption for their feature vectors. In our experiments, this algorithm runs in real time on QCIF (176×144) size videos on an Intel Pentium 200 MHz PC. Because this algorithm is a real time VO segmentation algorithm, we also discussed two direct applications of it, *i.e.*, real time VO generation for MPEG-4 codecs and content-based bit-rate control for traditional H.263 codecs.

Our work on automatic anchorperson detection is mainly designed for automatic broadcasting news video indexing and retrieval. To capture the consistent features of anchorpersons in various contexts, we proposed to model the anchorperson patterns with respect to their color and shape features. The detection problem is decomposed into a color-model based face region detection and a shape model based head-and-shoulder pattern detection problem. The statistical shape model design in this work is similar to what we used in the real time segmentation algorithm design for videoconference applications. However, in the offline scenario, shape model fitting is a much more expensive searching problem, which we solved with a downhill simplex searching algorithm.

Similarly, our work on face detection also proposed a combined color and texture based model design. Because video information in the multimedia world is always colored data, we have tried to merge the two types of face detection algorithms in the literature, *i.e.*, the skin-color based face detection and the texture based face

detection, in our work. Though each type of the algorithms has been discussed extensively in separate applications, our work has showed that they could be combined to generate better detection performance. In addition, our face models, both color and texture models are designed in the compressed domain of JPEG and MPEG rather than in the pixel domain. This work, especially the texture model design for face detection in the compressed domain is a new problem in the multimedia content analysis research. Our work discussed a number of fundamental problems in this area, such as block quantization problem, preprocessing in the DCT domain, and feature vector selection and classification in the DCT domain.

In semi-automatic analysis research, our major work is the design of an interactive authoring system for semi-automatic video object segmentation and annotation. This system features a new contour interpolation algorithm, which enables the user to define the contour of a video object on multiple frames while the computer interpolates the missing contours of this object on every frame automatically. Typical active contour (snake) model is adapted and the contour interpolation problem is decomposed into two directional contour tracking problems and a merging problem. In addition, new user interaction models are created for the user to interact with the computer. These include the *Interactive Rubberband* and the *Iterative Interpolation*. Experiments indicate that this system offers a good balance between algorithm complexity and user interaction efficiency.

Though this thesis work has solved a number of problems in video content analysis, based on the available results, many problems are raised for further research as well.

In automatic video analysis research, we believe application-oriented algorithm design will continue to be a major research direction in multimedia community. However, in order to obtain better performance, more efforts are needed to combine

the state-of-the-art techniques from areas of signal compression, computer vision and pattern recognition. An interesting problem worth mentioning is the face detection in the compressed DCT domain. Our work has shown that face detection directly based on the block based DCT parameters is feasible. However, to obtain better detection performance, more research work is necessary to investigate a better trade-off between model representation accuracy and representation efficiency. That is, it is not always good to create models in very high dimensional feature space with high representation accuracy. One problem for it is that longer feature vector means more computation power in model creating and fitting. Another is that quantization errors will become dominate in pattern feature variance in high dimensional feature space. Therefore, optimal determination of feature vector length is an important topic that influences the performance of the detection algorithms. In addition, better pattern classification tools such as neural networks and support vector machine are all good options to further improve the face detection performance.

Another interesting research topic for automatic video content analysis is to make use of multimedia information fusion. That is, in multimedia systems, content analysis should not only be confined to video information. Additional media forms, such as audio and text also offer useful information. An appropriate application example for that is automatic content analysis of broadcast news. Further improvement on our model-based anchorperson detection work could be achieved by incorporating works such as speech based speaker recognition and close caption based topic identification. Actually, one of our papers [67] has discussed some of the related problems.

In the near future, general-purpose VO segmentation research will mainly be based on semi-automatic system design. In order to make video objects popular, VO segmentation systems should have friendly user interfaces as well as highly effi-

cient processing algorithms. In this thesis work, our interactive system design uses contour based object representation, which, though convenient for multiple tracking information fusion, is not the only representation for general purpose VO segmentation. Other VO representation schemes, such as region-based representation and combined region and boundary-based representation are all interesting alternatives that should be studied for better interactive segmentation system design.

References

[1] T. Aach, A. Kaup, and R. Mester. Statistical model-based change detection in moving video. *Signal Processing*, 31:165–180, 1993.

[2] M. Abdel-Mottaleb and A. Elgammal. Face detection in complex environments from color images. In *IEEE International Conference on Image Processing*, Kobe, Japan, 1999.

[3] K. Aizawa. Human facial motion analysis and synthesis with application to model-based coding. In *Motion Analysis and Image Sequence Processing*. Kluwer Academic Publishers, MA., 1993.

[4] A.A. Alatan, E. Tuncel, and L. Onural. A rule-based method for object segmentation in video sequences. In *IEEE International Conference on Image Processing*, Santa Barbara, CA, 1997.

[5] A. Albiol, C.A. Bouman, and E.J. Delp. Face detection for pseudo-semantic labeling in video databases. In *IEEE International Conference on Image Processing*, Kobe, Japan, 1999.

[6] Y. Altunbask, P.E. Eren, and A.M. Tekalp. Region-based paraetric motion segmentation using color information. *Graphical Models and Image Processing*, 60(1):13–23, 1998.

[7] A.A. Amini, T.E. Weymouth, and R.C. Jain. Using dynamic programming for solving variational problems in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):885–867, 1990.

[8] H.S. Sawhney an d S. Ayer. Compact representations of videos through dominant and multiple motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8), 1996.

[9] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2), 1998.

[10] R. Castagno, T. Ebrahimi, and M. Kunt. Video segmentation based on multiple features for interactive multimedia applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(5):562–571, 1998.

[11] E. Chalom. Image segmentation using multi-dimensional attributes. Thesis Proposal, MIT Media Lab, 1996.

[12] E. Chalom and V.M. Bove Jr. Segmentation of an image sequence using multi-dimensional image attributes. In *IEEE International Conference on Image Processing*, Lausanne, Sept. 1996.

[13] S.-F. Chang, W. Chen, H.J. Meng, H. Sundaram, and D. Zhong. A fully automated content-based video search engine supporing spatiotemoral queries. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(5):602–615, 1998.

[14] R. Chellappa, C.L. Wilson, and S. Sirohey. Human and machine recognition of faces: A survey. *Proceedings of IEEE*, 83(5), 1995.

[15] L. Chiariglione. Statement on MPEG receiving the Emmy Award. MPEG 96, Oct. 2, 1996.

[16] T.C Chiueh, T. Mitra, and C.K. Yang. Zodiac: a history-based interactive video authoring system. In *ACM International Multimedia Conference*, Bristol, England, Sept. 1998.

[17] J.G. Choi, S.W. Lee, and S.D. Kim. Spatio-temporal video segmentation using a joint similarity measure. *IEEE Transactions on Circuits and Systems for Video Technology*, 7:279–285, 1997.

[18] R. Ciapini, L. Blanc-Feraud, M. Barlaud, and E. Salerno. Motion-based segmentation by means of active contours. In *IEEE International Conference on Image Processing*, Santa Barbara, CA, 1997.

[19] S. Colonnese, A. Neri, G. Russo, and P. Talone. Moving objects versus still background classification: a spatial temporal segmentation tool for MPEG-4. ISO/IEC JTC1/SC29/WG11, MPEG 96/571, 1996.

[20] T.F. Cootes, C.J. Taylor, D.H. Cooper, and J. Graham. Active shape models, their training and applications. *Computer Vision and Image Understanding*, 61:38–59, Jan. 1995.

[21] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*, chapter 25.2. MIT Press, 1990.

[22] P. Correia and F. Pereira. The role of analysis in content-based video coding and indexing. *Signal Processing*, 66:125–142, 1998.

[23] J. Dakss, S. Agamanolis, E. Chalom, and V.M. Bove. Hyperlinked video. In *Proceedings of the SPIE: Multimedia Systems and Applications*, 1998.

[24] F. Dell'Acqua and P. Gamba. Simplified modal analysis and search for reliable shape retrieval. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(5), 1998.

[25] Y. Deng and B.S. Manjunath. Content-based search of video using color, texture and motion. In *IEEE International Conference on Image Processing*, Santa Barbara, CA, 1997.

[26] W. Ding and B. Liu. Rate control of MPEG video coding and recording by rate-quantization modeling. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(1):12–19, Feb. 1996.

[27] R. Dugad and N. Ahuja. *A Fast scheme for image size change in the compressed domain*, April 1999. Manual under review.

[28] A. Eleftheriadis and A. Jacquin. Model-assisted coding of video telecoferencing sequences at low bit-rates. In *Proceedings IEEE International Symposium on Circuits and Systems*, pages 3.177–3.180, London, England, June 1994.

[29] A. Eleftheriadis and A. Jacquin. Automatic face location detection for model-assisted rate control in H.261-compatible coding of video. *Signal Processing: Image Communication*, 7:435–455, 1995.

[30] W. H. Press et al. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, 1993. second editon.

[31] A. Hanjalic *et al.* Template-based detection of anchorperson shots in news programs. In *IEEE International Conference on Image Processing*, Chicago, IL, 1998.

[32] H.J. Zhang *et al.* Automatic parsing and indexing of news video. *Multimedia Systems*, 2(6), 1995.

[33] M. Colobet *et al.* LISTEN: a system for locating and tracking individual speakers. In *Proceedings of the 2nd International Conference on Automatic Face and Gesture Recognition*, pages 283–288, Killington, VT, 1996.

[34] Q. Huang *et al.* Automated generation of news content hierarchy by integrating audio, video and text information. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, 1999.

[35] A.M. Ferman and A.M. Tekalp. Efficient filtering and clustering methods for temporal video segmentation and visual summarization. *Journal of Visual Communication and Image Representation*, 9(4):336–351, 4 1998.

[36] Y. Fu, A.T. Erdem, and A.M. Tekalp. Occlusion adaptive motion snake. In *IEEE International Conference on Image Processing*, Chicago, Oct. 1998.

[37] K. Fukunaga and W.L.G. Koontz. Application of the Karhunen-Loeve expansion to feature selection and ordering. *IEEE Transactions on Computers*, 19(4), 1970.

[38] D. Le Gall. MPEG: a video compression standard for multimedia applications. *Communcations of the ACM*, 34(4), 1991.

[39] C. Garcia and G. Tziritas. Face detection using quantized skin color regions and wavelet packet analysis. *IEEE Transactions on Multimedia*, 1(3), 1999.

[40] D. Geiger, A. Gupta, L.A. Costa, and J. Vlontzos. Dynamic programming for detecting, tracking and matching deformable contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(3):294–302, 1995.

[41] P. Gerken, M. Wollborn, and S. Schultz. An object-based layered codec as proposal for MPEG-4 scalability and compression tests - technical description. ISO/IEC JTC1/SC29/WG11, MPEG 95/359, 1995.

[42] C. Gu and M.C. Lee. Semantic video object tracking using region-based classification. In *IEEE International Conference on Image Processing*, Chicago, Oct. 1998.

[43] B. Gunsel and A.M. Tekalp. Similarity analysis for shape retrieval by example. In *Proceedings of International Conference on Computer Vision and Pattern Recognition*, 1996.

[44] B.G. Haskell, A. Puri, and A.N. Netravali. *Digital video: an introduction to MPEG-2.* Chapman and Hall, 1997.

[45] ITU Telecommunication Standardization Sector LBC-95, Study Group 15, Working Party 15/1, Expert's Group on Very Low Bitrate Visual Telephony. *VIDEO CODEC TEST MODEL TMN5*, Jan. 1995.

[46] A.K. Jain, Y. Zhong, and M.-P. Dubuisson-Jolly. Deformable template models: a review. *Signal Processing*, 71:109–129, 1998.

[47] A.K. Jain, Y. Zhong, and S. Lakshmanan. Object matching using deformable templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(3), 1996.

[48] ISO/IEC JTC1/SC29/WG11. Overview of the MPEG-4 version 1 standard, (N2078), 1998. MPEG 98, San Jose.

[49] ISO/IEC JTC1/SC29/WG11. MPEG-7 requirements document v. 10, (N2996), 1999. MPEG 99, Melbourne, Australia.

[50] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[51] C. Kervrann and F. Heitz. Statistical deformable model-based segmentation of image motion. *IEEE Transactions on Image Processing*, 8(4), 1999.

[52] J.B. Lee and A. Eleftheriadis. Motion adaptive model-assisted compatible coding with spatio-temporal scalability. In *Proceedings of SPIE Visual Communication and Image Processing*, pages 622–634, Feb. 1997.

[53] J.B. Lee and B.G. Lee. Transform domain filtering based on piplining structure. *IEEE Transactions on Signal Processing*, 40:2061–2064, 1992.

[54] Y. Lee, F. Kossentini, R. Ward, and M. Smith. Towards MPEG4: An improved H.263-based video coder. *Signal Processing: Image Communication*, 10:143–158, 1997.

[55] C. Lettera and L. Mastera. Foreground/background segmentation in videotelephony. *Signal Processing: Image Communication*, 1:181–189, 1991.

[56] M. Lew. Information theoretic view-based and modular face detection. In *Proceedings of the 2nd International Conference on Automatic Face and Gesture Recognition*, pages 198–203, Killington, VT, 1996.

[57] H. Li, A. Lundmark, and R. Forchheimer. Image sequence coding at very low bit rate: A review. *IEEE Transactions on Image Processing*, 3(5):589–608, 1994.

[58] H. Li, P. Roivainen, and R. Forchheimer. 3D motion estimation in model-based facial image coding. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 15(6), June 1993.

[59] I-J. Lin and S.Y. Kung. Circular viterbi based adaptive system for automatic video object segmentation. In *IEEE Workshop on Multimedia Signal Processing*, Los Angleles, 1998.

[60] Y.T. Lin and Y.L. Chang. Tracking deformable objects with the active contour model. In *IEEE International Conference On Multimedia Computing and Systems*, Ottawa, Canada, June 1997.

[61] Z. Liu and Q. Huang. Detecting news reporting using audio/visual information. In *IEEE International Conference on Image Processing*, Kobe, Japan, 1999.

[62] Z. Liu and Q. Huang. Adaptive anchor detection using online trained audio/visual model. In *Proceedings of SPIE: Storage and Retrieval for Media Databases*, volume 3972, 2000.

[63] H. Luo and A. Eleftheriadis. Designing an interactive tool for video object segmentation and annotation. In *ACM International Multimedia Conference*, pages 477–487, Orlando, FL, 1999.

[64] H. Luo and A. Eleftheriadis. Spatial/temporal bi-directional active contour tracking for semi-automatic video labeling and vo generation. In *IEEE International Conference on Image Processing*, Kobe, Japan, 1999.

[65] H. Luo and A. Eleftheriadis. On face detection in the compressed domain. In *ACM International Multimedia Conference*, Los Angles, CA, 2000. To appear.

[66] H. Luo and A. Eleftheriadis. Statistical model based video segmentation and its application to very low bit rate video coding. *Signal Processing: Image Communication*, 2000. To appear.

[67] H. Luo and Q. Huang. Experiments on automatic categorization of broadcasting news. Technical report, AT&T Research Labs, 1999. Also to be submitted to International Conference on Spoken Language Processing (ICSLP) 2000, Beijing, Oct. 2000.

[68] H. Luo, J. Koloherious, and A. Eleftheriadis. Statistical model based video segmentation and its application to very low bit rate video coding. In *IEEE International Conference on Image Processing*, Chicago, IL, 1998.

[69] H. Luo and Y.J. Zhang. A system for image segmentation evaluation. *Journal of Data Acquisition and Processing*, 12(1):18–22, 1997.

[70] H. Luo and Y.J. Zhang. Optimal selection of segmentation algorithms based on performance evaluation. *Optical Engineering*, 39(6):1450–1456, 2000.

[71] A.-R. Mansouri and J. Konrad. Motion segmentation with level sets. In *IEEE International Conference on Image Processing*, Kobe, Japan, 1999.

[72] A. Martelli. A heuristic search methods to edge and contour detection. *Communication of ACM*, 19(2):73–83, 1976.

[73] S. McKenna and S. Gong. Tracking faces. In *Proceedings of the 2nd International Conference on Automatic Face and Gesture Recognition*, pages 271–275, Killington, VT, 1996.

[74] J. Meng and S.-F. Chang. Scene change detection in a mpeg compressed video sequence. In *Proceedings of SPIE, Symposium on Electronic Imaging: Science and Technology (EI'95) - Storage & Retrieval for Image and Video Databases IV*, volume 2417, San Jose, CA, 1995.

[75] J. Meng and S.-F. Chang. Tools for compressed-domain video indexing and editing. In *Proceedings of SPIE, Symposium on Electronic Imaging: Science and Technology (EI'96) - Storage & Retrieval for Image and Video Databases IV*, volume 2670, San Jose, CA, 1996.

[76] B. Menser and F. Muller. Face detection in color images using principle components analysis. In *Seventh International Conference on Image Processing and Applications*, volume 2, pages 620–624, 1999.

[77] N. Merhav and V. Bhaskaran. Fast algorithms for DCT-domain image downsampling and for inverse motion compensation. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(3), 1997.

[78] B. Moghadda and A. Pentland. Probalilistic visual learning for object detection. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition*, pages 786–793, Cambridge, MA, 1995.

[79] B. Moghaddam and A. Pentland. An automatic system for model-based coding of faces. In *IEEE Data Compression Conference*, Snowbird, Utah, March 1995.

[80] E. Mortensen and W. Barrett. Intelligent scissors for image composition. In *Proceedings of ACM SIGGRAPH 95*, pages 191–198, Los Angeles, CA, 1995.

[81] E.N. Mortensen and W.A. Barrett. Interactive segmentation with intelligent scissors. *Graphical Models and Image Processing*, 60:349–384, 1998.

[82] S. Mukhopadhyay and B. Smith. Passive capture and structuring of lectures. In *ACM International Multimedia Conference*, pages 477–487, Orlando, FL, 1999.

[83] A.V. Nefian and M.H. Hayes III. Face detection and recognition using hidden markov models. In *IEEE International Conference on Image Processing*, Chicago, IL, 1998.

[84] J. Ohm and P. Ma. Feature-based cluster segmentation of image sequences. In *IEEE International Conference on Image Processing*, Santa Barbara, CA, 1997.

[85] N. Oliver, A. Pentland, and F. Berard. LAFTER: A real-time lips and face tracker with facial expression recognition. In *Proceedings of International Conference on Computer Vision and Pattern Recognition*, S.Juan, Puerto Rico, June 1997.

[86] E.E. Osuna, R. Freund, and F. Girosi. Support vector machines: training and applications. Technical report, MIT AI Lab, March 1997. A.I.Memo 1602.

[87] A. Puri and A. Eleftheriadis. MPEG-4: An object-based multimedia coding standard supporting mobile applications. *ACM Mobile Networks and Applications*, 1997.

[88] H.A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 20(1), 1998.

[89] E. Sahouria and A. Zakhor. Motion indexing of video. In *IEEE International Conference on Image Processing*, Santa Barbara, CA, 1997.

[90] P. Salembier and M. Pardas. Hierarchical morphological segmentation for image sequence coding. *IEEE Transactions on Image Processing*, 3:639–651, Sept. 1994.

[91] S. Satoh, Y. Nakamura, and T. Kanade. Name-It: naming and detecting faces in news videos. *IEEE Multimedia Magazing*, pages 22–35, 1999.

[92] S. Sclaroff. Deformable prototypes for encoding shape categories in image databases. *Pattern Recognition*, 30(4), April 1997.

[93] J. R. Smith and S.-F. Chang. Transform features for texture classification and discrimination in large image databases. In *IEEE International Conference on Image Processing*, Austin, TX, November 1994.

[94] J. R. Smith and S.-F. Chang. Tools and techniques for color image retrieval. In *Storage and Retrieval for Image and Video Databases IV, IS&T/SPIE Electronic Imaging: Science & Technology 96*, volume 2670, San Jose, CA, February 1996.

[95] K. Sobottka and I. Pitas. Segmentation and tracking of faces in color images. In *Proceedings of 2nd International Conference on Automatic Face and Gesture Recognition*, pages 236–241, Killington, Vermont, 1996.

[96] E. Steinbach, P. Eisert, and B. Girod. Motion-based analysis and segmentation of image sequences using 3-d scene model. *Signal Processing*, 66:233–247, 1998.

[97] K.K. Sung. *Learning and Example Selection for Object and Pattern Detection.* PhD thesis, MIT AI lab, 1996. also available as MIT tech. Report AITR 1572.

[98] B. Tao, K. Robbins, and B. Dickinson. Image retrieval with templates of arbitrary size. In *Proceedings of SPIE: Storage and Retrieval for Image and Video database V*, 1997.

[99] A.M. Tekalp. *Digital Video Processing.* Prentice Hall, 1996.

[100] Telenor. H.263 software TMN2.0. (ftp://bonde.nta.no/pub/tmn/software/tmn-2.0.tar.gz).

[101] S. Theodoridis and K. Koutroumbas. *Pattern Recognition.* Academic Press, San Diego, 1999.

[102] E. Viscito and C. Gonzales. A video compression algorithm with adaptive bit-allocation and quantization. In *Proceedings of SPIE: Visual Commun. and Image Processing 91*, volume 1605, pages 205–216, Boston, MA., Nov. 1991.

[103] G.K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4), 1991.

[104] D. Wang. Unsupervised video segmentation based on watersheds and temporal tracking. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(5):539–547, Sept. 1998.

[105] H. Wang and S.-F. Chang. A highly efficient system for automatic face region detection in mpeg video. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(4), 1997.

[106] J. Wang and E.H. Adelson. Representing moving images with layers. *IEEE Transactions on Image Processing*, 3(5), 1994.

[107] J. Wang and E.H. Adelson. Spatio-temporal segmentation of video data. In *Proceedings of the SPIE: Image and Video Processing II*, volume 2182, San Jose, 1994.

[108] J. Webb and K. Oehler. A simple rate-distortion model parameter estimation, and application to real-time rate control for DCT-based coders. In *IEEE International Conference on Image Processing*, Santa Barbara, CA., Oct. 1997.

[109] IBM HotVideo website. http://www.research.ibm.com/topics/innovate /multimedia/.

[110] Veon website. http://www.veon.com/.

[111] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder, real-time tracking of the human body. Technical Report 353, MIT Media Lab, 1995. also in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 7, pp. 780-785, July 1997.

[112] M.-H. Yang and N. Ahuja. Detecting human faces in color images. In *IEEE International Conference on Image Processing*, Chicago, IL, 1998.

[113] M.-H. Yang and N. Ahuja. Face detection using a mixture of factor analyzers. In *IEEE International Conference on Image Processing*, Kobe, Japan, 1999.

[114] M.M. Yeung, B.-L. Yeo, and B. Liu. Segmentation of video by clustering and graph analysis. *Computer Vision and and Image Understanding*, June 1998.

[115] Y.J. Zhang. Evaluation and comparison of different segmentation algorithms. *Pattern Recognition Letters*, 18:963–974, 1997.

[116] D. Zhong and S.-F. Chang. An integrated approach for content-based video object segmentation and retrieval. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 1259–1268, Dec 1999.

[117] D. Zhong and S.F. Chang. AMOS: an active system for MPEG-4 video object segmentation. In *IEEE International Conference on Image Processing*, Chicago, Oct. 1998.

[118] D. Zhong, H.J. Zhang, and S.-F. Chang. Clusterging methods for video browsing and annotation. In *Storage and Retrieval for Image and Video Databases IV, IS&T/SPIE Electronic Imaging: Science & Technology 96*, 1996.

[119] X. Zhou, Y. Rui, and T.S. Huang. Water-filling: A novel way for image structural feature extraction. In *IEEE International Conference on Image Processing*, Kobe, Japan, October 1999.