Feature-based Video Sequence Identification

Kok Meng Pua

B.Sc. (CoE), University of Kansas, 1993
M.Sc. (EE), University of Kansas, 1995

Submitted to the Department of Electrical Engineering and Computer Science and the Faculty of the Graduate School of the University of Kansas in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis Cmmittee:

Chairman

Date dissertation defended: May/17/02

Abstract

This dissertation reports on research, development, and evaluation of a color based video sequence identification and tracking algorithm. We describe an automatic video sequence identification and tracking algorithm that detects and extracts repeated video sequences from a continuous video stream. Because our technique is domain and video source independent, it is applicable to any video stream that is repetitive and changes slowly over time.

We digitize and segment a continuous video stream into video sequences using color histogram-based techniques. Our video sequence identification approach groups individual frames together based on their color features. In particular, we use the nine color moments, namely the mean, variance, and skew of each primary color component of the RBG color format. Then, we identify similar video sequences based on how many similar frames they contain. Finally, we compare the similar video sequences frame by frame to identify repeated video sequences

We studied the efficiency and the effectiveness of our algorithms on 24 or more hours of video data from two different sources. We found that the technique accurately identifies repeated sequences, producing recall and precision values both over 90%. We also evaluated the applicability of our technique as a lossless compression algorithm. By removing repeated sequences from the video archive, we achieve a compression gain ratio of 30% on each source.

Acknowledgements

I would like to express my deepest gratitude to my advisor and mentor, Dr. Susan Gauch, for her invaluable guidance and unwavering support throughout my education at the University of Kansas. Without her encouragement and patience, this work can not be done. I would also like to extent my gratitude to Dr. Joseph Evans, Dr. Jerry James, and Dr. Tom Schreiber for serving on my thesis committee and their encouragement.

I would like to thank Dr. John Gauch, who is also serving on my thesis committee, for valuable discussions and information sharing on issues of the image and video processing**.**

Finally, I want to express my gratitude and love to my wife, Mei Mei Fong, for her loving support and encouragement.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

1.1 Driving Problem

One of the main technology achievements in the mid-twentieth century was the invention of the television set and hence the proliferation of video signal broadcasting. In the late 1990s, we saw the technology of the late-twentieth century, the Internet, used as a new broadcasting media, bringing video streams to personal computers (PCs). Due to the combination of these two technologies, we are able to see the latest events and stories happening around the world in real time by watching television or video streams over the Internet. The easy availability of these media transmissions has created repeated broadcast content that causes the ineffective use of a viewer's time and inefficient use of storage media in archives. Unlike earlier times when there were only a few television channels, cable networks and Internet web sites provide continuous program coverage of news and documentaries. However, if we

watch the same channel at different times of the day to keep abreast of television programs, we can spend hours only to discover that a large portion of the programs and video sequences are repeated from the previous television program sessions.

What is needed is a video sequence tracking system that could combine, for instance, a day's worth of program coverage into one shorter program session containing only the unique stories. In other words, we need an effective way to shrink the length of a viewer's television program session without losing any content. One possibility is to compare continuous program coverage, remove repeated video sequences and combine all and only the unique video. This should create a more compact representation of the entire days. Shorter television program sessions could lead to shorter broadcasting time and better bandwidth utilization. Viewers could use the time saved to receive content from a wider variety of programs.

1.2 Project Goals of the Thesis

This dissertation describes an automatic video sequence identification and tracking technique. The focus of the work is to design, implement, and test algorithms and technologies to extract unique video sequences from repetitive continuous video streams. Our technique is domain and video source independent so that it can be used on any video stream that contains repeated sequences. For testing and evaluation purposes, we have applied this technology to two television channel video streams.

Our work has three main goals: 1) design and implement an efficient real time video sequence identification and tracking technique; 2) reduce the storage requirements of video archive built from the stream by omitting repeated sequences; and 3) reduce the amount of time a user needs to view a program by displaying only new, unique material. We evaluated the accuracy of our video sequence identification and tracking techniques using a collection of 32 hours of continuous video. We evaluated the compression achievable in both the user's viewing time and the archive storage requirements. We have also evaluated how well the technique performs on other video sources by testing the technique on a 24 hours of video stream collected from a different television source.

1.3 The Broader Picture

This work could form one component of a video archive system for content-based topic tracking. Figure 1 illustrates a function block diagram for a fully automated, content-based story tracking system for television news programs. The question now is: Is it feasible to build such a fully automated video content-based news story tracking system that can represent continuous news coverage as a series of unique topics/stories to achieve better content and compression efficiency?

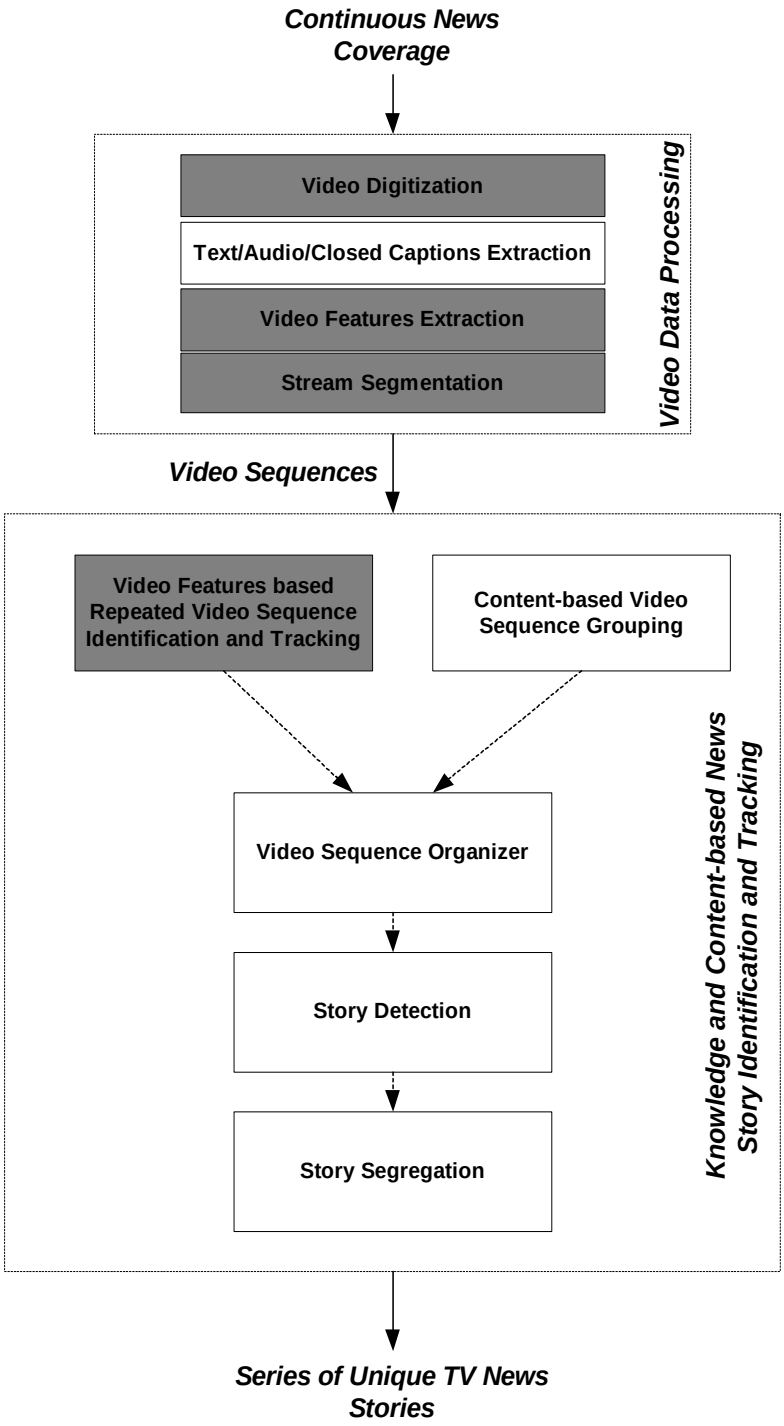


Figure 1.1 Functional Block Diagram of An Ideal Television News Topic Tracking System

Given the current video signal processing technologies, without any human intervention and manual editing, the solution to the above problem is still not achievable. The fast growth of multimedia information in image and video databases in the last ten years has triggered research on efficient video retrieval and processing methods, but none of these projects concentrate on finding techniques for identifying unique video sequences and tracking news stories from a given video source. In order to detect and aggregate news stories from a video source, we need a fully automated video signal processing system that can first apply content-based video processing techniques to track and extract all the unique video sequences from the input source. Then, video sequences must be grouped into different stories using video abstractions such as closed captions, audio and video features. The research presented in this dissertation has tried to answer some (indicated as grayed blocks in Figure 1.1), although not all, of these problems. In particular, we concentrate on the feature-based identification and tracking of unique video sequences, but not on the knowledge-intensive activity of content-based aggregation into stories.

Chapter 2

The Related Work

This research has its roots in image and video processing techniques and in topic detection and tracking of text documents. Related digital image and video processing problems include the abstraction and representation of a video content, the detection and segmentation of the video shots and scenes, and the content-based video archive retrieval and storage. Other ongoing areas of video processing research study video similarity measurement, topical event detection in video sources, commercial detection, and story-based video segmentation. The following sections discuss some of this related work.

2.1 Image Abstraction and Similarity Measure

A video sequence is made up of number of video frames captured in temporal order in which each video frame can be treated as one single image. Hence, a video sequence can be seen as an ordered stream of images with each image represented by its own abstraction. Existing research on content-based image retrieval explores multiple ways of representing these images for content-based similarity measurement and retrieval. Images can be represented by properties of color, shape, and edge features. One of the most popular ways of representing an image is to use its color histogram. This feature-based image representation has been shown to be efficient and effective in the content-based image retrieval [11][24][28]. A color histogram describes the global color distribution in an image. The color histogram is extremely easy to compute and insensitive to small changes in viewing positions and partial occlusion. The degree of similarity between two represented images is calculated as the distance between two color histograms. However, a color histogram only records an image's overall color composition, so images with very different appearances can have similar color histograms. Pan and Zabih [26] show that the histogram method is not robust to large appearance changes and is liable to produce false positives due to the lack of any spatial information.

Several approaches have attempted to incorporate spatial information with color. Smith and Chang [36] propose dividing images into sub-regions and imposing positional constraints on the image comparison (image partitioning). In their research, an image is partitioned into binary color sets. The binary color sets,

calculated using histogram back-projection [38], and their location information constitute the feature for an image. This feature can be used to perform region-based queries. Stricker and Dimai [37] divide an image into five fixed overlapping regions and extract the first three color moments of each region to form a feature vector for the image. The storage requirements for this method are very low. The use of overlapping regions makes the feature vectors relatively insensitive to small rotations or translations. Pass and Zabih [27] partition histogram bins by the spatial coherence of pixels. In their work, a pixel is coherent if it is a part of some "sizable" similar-colored region, and incoherent otherwise. A color coherence vector (CCV) is created to represent this classification for each color in the image. CCVs are fast to compute and appear to perform better than histograms. The notion of CCV is also extended in [27] using additional features to further refine the CCV-refined histogram.

Since histogram refinement methods depend on local properties, they are unlikely to tolerate large image appearance changes. The same problem occurs in the image partitioning approach that depends on pixel position. The correlogram method proposed by Huang and Kumar [18][19] takes into account the local spatial correlation between colors as well as the global distribution of this spatial correlation. A color correlogram of an image is a table indexed by color pair, where the k-th entry for *(i, j)* specifies the probability of finding a pixel of color j at a distance k from a pixel of color *i* in the image. The correlogram is easy to compute and the size of the feature is fairly small. It has also been shown to be robust to large image changes.

Other image features used for image similarity comparison are the shape and invariant properties of the color image. Geusebroek and Koelma [17] demonstrate an image retrieval system based on local color invariants. For each image stored in the database, color edge invariants for shadow and highlights are extracted. Shape invariant descriptors are computed from the edge map and the resulting shape features are used to index the image. Swain and Ballard [38] use dominant colors to construct an approximate representation of color distribution of a image and the results have shown that using only a few dominant colors will not degrade the performance of color image matching. In fact, since small histogram bins are likely to be the result of noise, performance may even be enhanced. Rowe and Boreczky [34] represent an image by the first three color moments for each color component and experimental evidence has shown image similarity based on color moments is more robust than that based on color histograms.

In another approach, Kato and Zhang [22][42] derive edges from an image using a technique such as Sobel filter to provide good cues for content. Two images can be then compared for similarity measure by calculating a correlation between their edge maps. However, these comparison methods are limited by their dependency on image resolution, size and orientation.

2.2 Video Sequence Abstraction and Feature based Video Similarity Measure

The ease of capturing and encoding digital video has created the need for new technologies able to handle multimedia information. One of the basic video processing techniques needed to handle video is the representation of video sequences. Typically, the video source is segmented into either shots or scenes. A video shot is defined as a continuous roll of the camera while a video scene is a collection of shots that occur in a single location or are temporally unified. Thus, a scene is a sequence of video shots representing continuous action. In our research, we define a video sequence as a video shot. One way of abstracting a video sequence is to map the entire video segment to some small number of representative images, usually called key frames [40][42][43]. Key frames are still images that best represent the content of the video sequence. They may be either extracted or constructed from the original video data. Index and video features can be constructed from these key-frames using image abstraction techniques discussed in the previous section to provide a key-frame similarity measure between two video sequences. Zhang and Low [42] represent video sequences based on key-frame color, texture, shape, and edge features.

Instead of selecting still images as key-frames for video sequences, Arman [3] represents each video sequence using a representative frame called an Rframe. Each

Rframe consists of a body, which is the 10^{th} frame of the video sequence, four motion-tracking regions, and the video sequence length indicator. The shape and color properties of these Rframes are calculated and used to measure similarity between Rframes, and hence their respective video sequences. The shape property of an Rframe is represented using the moment invariant while its color property is represented by the color histogram. The output of the moment-based and color histogram-based analyses are two floating numbers describing the similarity in shape and in color of the Rframes' body. A mapping function is used to map both entities onto a common space in order to combine and compare these two different entities.

Chueng and Zakhor [5] consider the use of meta-data and video domain methods to detect similar videos on the web. In their work, meta-data is extracted from the textual and hyperlink information associated with each video sequence while in the video domain a video signature is created for video similarity measure. In the meta-data method, each video sequence is represented by a set consisting of all the distinct terms found in the associated meta-data. The degree of meta-similarity, which determines the degree of video sequence similarity, is defined as the ratio between the size of the intersection and the union of the two meta-data sets. The video signature for each video sequence is constructed by selecting a small number of frames that are most similar to a set of random seed images. In the video signature method, the similarity between video sequences is based solely on the similarity between individual signature frames selected. The degree of signature frame similarity is determined by measuring visual feature distance between frames. Since

it needs to be done for every seed and every pair of video in the database, they propose a statistical pruning procedure [6] to the complexity of the frame distance computation. Also, a new signature clustering algorithm [6] is proposed to further improve similar video sequence retrieval performance by providing an efficient organization of data that allows users to focus on relevant information. This clustering algorithm treats all the signatures as an abstract threshold graph, where the threshold is determined based on local data statistics. The experimental results show that this algorithm outperforms the simple thresholding and hierarchical clustering techniques proposed by Chueng [5].

The VisualGREP project [18] includes a systematic method to compare and retrieve video sequences at four levels of temporal resolution: frame, shot, scene and video. At each level, features are employed to transform the video sequences into an appropriate representation. Features used in this system are color, motion intensity and frontal faces. A normalized measure of distance between the representations of two video sequences is defined to capture their similarity. At the frame level, frames are compared by any image feature. At shot levels, the image features derived from their respective frames represent shots and hence the similarity is determined by frame values and also the temporal order. A scene is represented by multiple shots and hence they are compared based on the concepts developed for shots, resulting in a recursive computation scheme. The same recursive computation scheme applies to the video level. The system allows users to easily adjust a feature's distance measure to their actual desired similarity judgment. The method presented is capable of

comparing temporally large entities such as scenes and full-length feature films for general video.

Dimitiova Abdel-Mottaleb [8][9] regards the average distance of all the corresponding frames as two videos' similarity and defines that video frame sequence must obey temporal order. He introduces a novel approach for video similarity and retrieval from a large archive of MPEG compressed video clips. The proposed method takes a video clips as a query and searches the database for clips with similar contents characterized by a sequence of representative frames signatures constructed from the Discrete Cosine coefficient and motion information. In contrast, Wu [40] combines color and textual features from key-frames for shot similarity measure. The color feature is defined by histogram in HIS color space, represented as a 32-floating point number. The texture feature is composed of three floating point: coarseness, contrast and direction. On the whole, visual features of the key-frame are expressed as a vector of 35-dimension. The Euclidean distance between two vectors and the shot duration determines the degree of similarity between two shots represented by these key-frames. The video similarity measure takes into account the temporal order of similar shots and the number of scattered shots that cannot find a similar counterpart to measure the final degree of similarity between two video sequences.

The work by Rui and Huang [35] show that the similarity of two shots is an increasing function of visual similarity and a decreasing function of shot size difference. Visual similarity is measured based on the shots' spatial and temporal features. Their current algorithm uses the color histogram for the first and last frames

as the spatial feature for the shot. The temporal feature is represented by calculating the average of the color histogram difference between adjacent frames in the shot.

The VidWatch Project [12] demonstrates the use of the first three color moments of the red, green, and blue color components to represent a video frame. Thus, each video frame is represented by a vector of nine floating point numbers and hence a video sequence is represented by multiple vectors of nine, one for each video frame. Two video streams are compared by measuring the sum of the absolute moment difference of video frames represented by the nine color moments. The VidWatch project uses the same video frame abstraction and video sequence comparison method to detect commercial replays on a television channel. The results from both applications show that the color moment technique can be robustly deployed to represent a large video archive. The feature storage requirements are small, yet the nine color moment values are able to represent each video frame uniquely and hence identify duplicate (or different) video sequences.

The goal of our video identification technique is to detect and identify repeated video sequences from video archive captured from a continuous video stream of a television channel. We have developed a sequential video frame comparison technique that compares each video frame from two video sequences in temporal order to decide if they are repeated or not. Since we need to select a method of video frame feature abstraction that requires little data storage while creating a unique abstraction for each video frame, we use the color moment feature proposed in [12] as our video sequence abstraction.

2.3 Topical Video Event Detection

Since they do attempt to model the semantic content of the video, the feature-based video indexing, retrieval and similarity measure methods may not necessarily be semantically meaningful or relevant. A more advanced video content analysis method that is semantically meaningful is needed to more effectively identify and label video content and help users find what they are looking for. The detection of semantic events within video streams presents a new research area in content-based video processing. The goal of event-based video detection methods proposed in and implemented in [7][30][39][41] are to visually and semantically describe the content of video so that it is meaningful and significant to viewers. Qian and Haering [30] design an event-based video indexing, summarization and browsing for animal hunt detection in wildlife documentaries. Texture, color, and motion features are extracted and motion blobs are detected. A neural network is employed to verify whether the motion blobs belong to objects of interest. Shot summaries are generated and are used to detect video segments that contain events of interest.

The SmartWatch Project in [7] combines the use of textual (closed caption or transcripts) and aural analysis to automatically detect truly "interesting" events in video sequences. Tanveer and Srinivasan [39] use the image content of foils to detect visual events in which the foil is displayed and captured in the video stream. The textual phrases listed on a foil are used as an indication of a topic events, the audio track is analyzed to detect where the best evidence for the topical phrases is heard. The combined results of the visual and audio event detection determine the time

occurrence of the video event. Yoon and DeMenthon [41] describe the use of motion vectors to detect interesting dynamic events based on the information in the compressed domain. Their method takes advantage of motion encoding without the need for full frame decompression, and hence their approach has a lower computation cost.

Instead of video sequence organization and detection, the Topic Detection and Tracking Project (TDT) [1][10] sponsored by NIST aims to develop technologies for retrieval and automatic organization of text and speech information such as news coverage on television and radio. The purpose of this project is to advance the state of the art in technologies required to segment, detect and track topical information in a stream consisting of both text speeches from newswire, radio and television news broadcast programs. Assuming the presence of textual information which can be used to semantically abstract the content of it video source, technologies for topic and event detection on text documents developed in TDT project can be extended for the purpose of topic and event detection from video sources.

One common requirement of all the event based video detection techniques described above is that they are heavily dependent on specific artifacts and are domain specific. This limits their effectiveness and applicability in different domains. An ideal model of fully automated video content-based event or topic tracking method will be an extensible computational approach that may be adapted to detect different events in different domains. While we do not attempt to track individual video stories through time, the first step in this process is the identification

of new content. Thus, our work can be considered a first step towards domain independent topic and tracking for video sources. In particular, we concentrate on feature-based identification of video sequences, but not the knowledge-intensive activity of content-based aggregation into stories or events.

Chapter 3

Pilot Work

3.1 The VISION Digital Video Library

The acronym VISION stands for "Video Indexing and Searching Over the Networks." It is a system developed at the University of Kansas as a testbed for evaluating automatic and comprehensive mechanisms for library creation and content-based search, retrieval, filtering and browsing of video across networks with a wide range of bandwidths [14][15]. The pilot system was populated with a collection of news videos from CNN [16]. These videos were automatically partitioned into story segments based on their content and stored in a multimedia database. A client-server based graphical user interface was developed to enable users to remotely search this library and view selected video segments over networks of different bandwidths. Additionally, VISION classifies the incoming videos with respect to a taxonomy of categories and will selectively send users videos which

match their individual profiles. The archive can also be explored by browsing through the taxonomy.

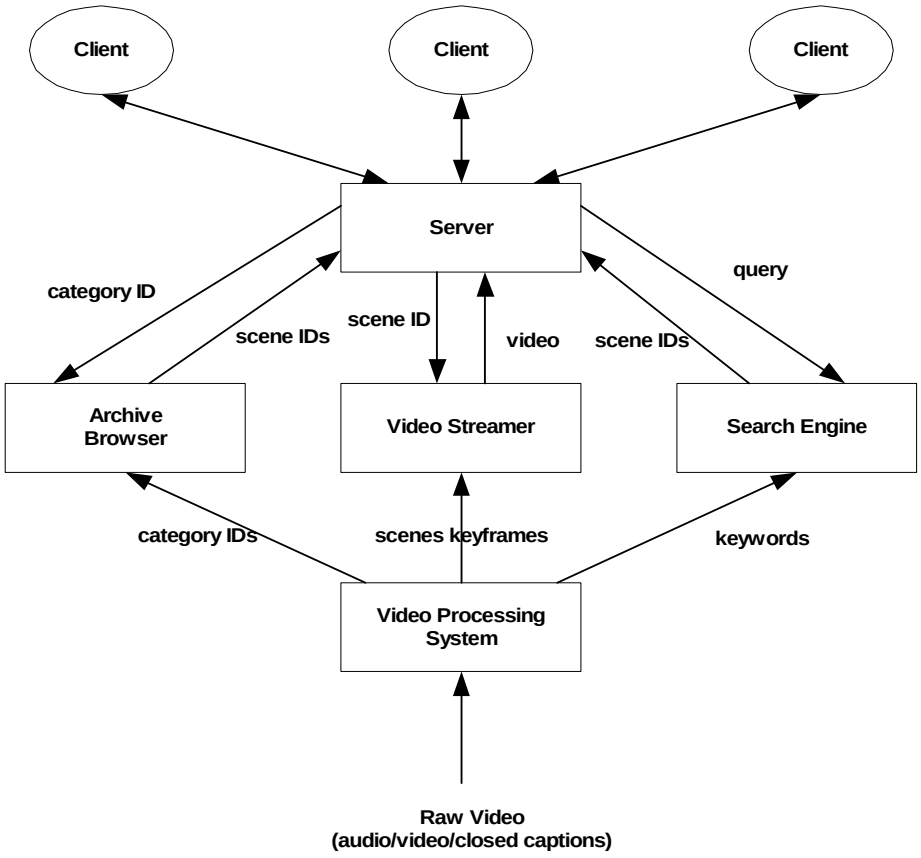


Figure 3.1 The architecture of the VISION system

The architecture of the first version of VISION is summarized in Figure 3.1. Although we originally developed our own client, server, and video streamer, the later VISION system uses a World Wide Web server and an Internet browser for the Server and Client, and the RealMedia Server and Client for video streaming. The Search Engine is an implementation of the vector space model and the Archive Browser is a cgi program which presents a browsable hierarchy of concepts (the

taxonomy) or a clustered set of keyframes to the user. The Video Processing System can continuously capture, segment, compress, classify, extract keyframes (and their features) and store and index video clips from a live broadcast feed in real-time.

The VISION project has also been extended to support real-time video scene detection and segmentation. A pipelined digital video processing architecture was developed that is capable of digitizing, processing, indexing and compressing video in real time on an inexpensive general purpose computer [13]. It uses a three-phase segmentation algorithm that combines video, audio and closed captions to detect scene changes. The videos were automatically partitioned into short scenes using combination of video, audio and closed-caption information and the resulting scenes are indexed based on their captions and stored in a multimedia database. An image feature based algorithm produces over-segmentation because it detects camera shots not just scene changes. Higher levels of video representation such as closed caption and audio are used to merge some of these shots so the resulting scenes will be more semantically unified. The studies of the effect of closed-caption based merging used after image-based video segmentation in [32] shows that the method significantly reduces the over-segmentation phenomenon and improves the accuracy of scene detection.

3.2 The VIDSEEK Project

One of the goals of the VISION project was to develop a client-server-based graphical user interface to enable users to remotely search the video archive and view

selected video segments over networks of different bandwidths. The VIDSEEK project [29] was designed to complement this goal by developing a web-enabled digital video library browsing system. With the explosive growth of information available in the World Wide Web, most queries result in many retrieved documents only some of which are relevant. Accessing digital video information is an even harder problem because content-based video indexing is difficult and the volume of retrieved video data is enormous. VIDSEEK (Figure 2.1) is a dynamic Web-based digital video library browsing system that allows users to preview the contents of the VISION digital video library via automatically selected and organized key frames. The focus of this system is the dynamic organization, i.e., categorization and clustering, of the video abstractions to provide a sophisticated tool for video archive exploration.

The system supports two main features, namely dynamic clustering-on-demand and category-based browsing. The dynamic clustering-on-demand allows users organize the digital video library clips into clusters based on multiple user-specified video features. The category-based browsing allows users to interactively and dynamically filter the VISION digital video library clips based on a given set of constraints, such as video source, keywords, and date of capture. This hybrid of browsing and searching system provides a powerful and flexible video archive exploration tool. The need for video clip playback can be reduced by allowing users to browse through video abstractions such as multiple key frames, category and caption information which provide a summary of video clip content.

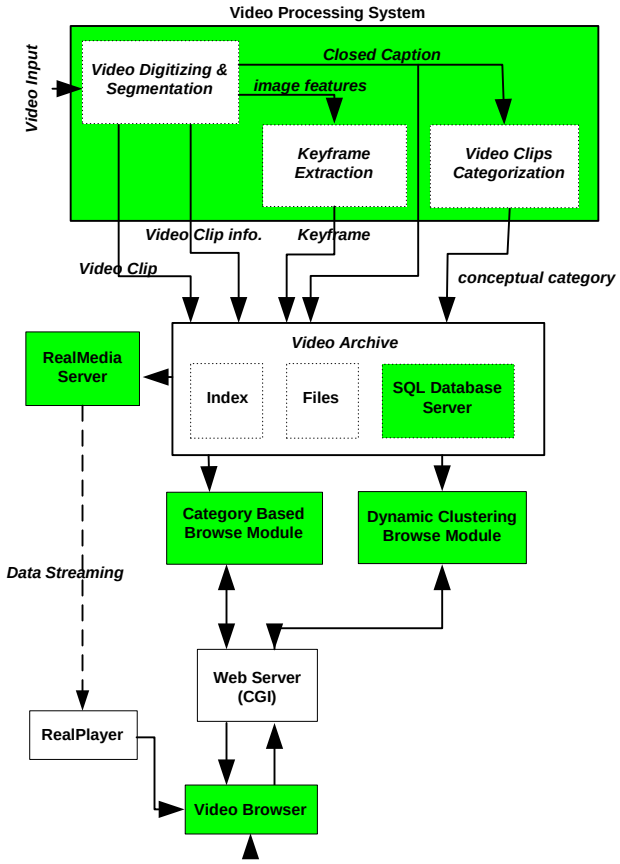


Figure 3.2 The system block diagram of VIDSEEK Browsing System

3.3 The VIDWATCH Project

Whenever video content is licensed and broadcast by distributors such as cable operators, the producers and owners of that content need to verify that their video is reaching customers correctly. The goal of the VidWatch [12] project is to develop methods to transmit and compare video features from two or more video streams in real-time to determine if the video broadcasts are reaching customers unchanged in each distributor's market. In other words, the research group developed content-based video analysis methods to provide video authentication. One of the main achievements of this research work was the patented automated video information

processing technique using features of color moments to characterize video content continuously in real-time.

The VidWatch project utilizes a client-server architecture to provide 24 by 7 video content analysis and video authentication over a wide area network of PCs with video digitizers. Whenever video differences are detected, the system digitizes and encodes the broadcast video and the customer video, and uses this information to generates daily video authentication reports for each channel and location being monitored. The VidWatch product has been deployed and successfully field tested for three years in cooperation with a major international television broadcasting network company.

This underlying video authentication technology can be used to compare any two video streams that are being simultaneously transmitted to different locations. For example, VidWatch could be used to monitor the "transmission quality" of digital video sources being streamed over the Internet in any format. Alternatively, the differences detected by VidWatch could be used to boost or correct a distorted video signal.

Chapter 4

Overview of Video Sequence Identification and Tracking System

4.1 The Approach

The process flow of the Video Sequence Identification and Tracking System has four main processes (Figure 4.1). First, during video processing and segmentation, we extract video abstractions and segment the video stream into multiple video sequences. Then, using video sequence hashing process, we identify similar video sequences from the stream of input video sequences. Next, we compare the input video sequences to any similar video sequences frame by frame to decide whether or not the input video sequence is truly a repeat or if it is a new sequence. Finally, we record temporal information for the video sequence that can be used to track video sequence occurrences or to reproduce the original video stream. An overview of the processes is given in this chapter and they are described in detail in Chapter 5.

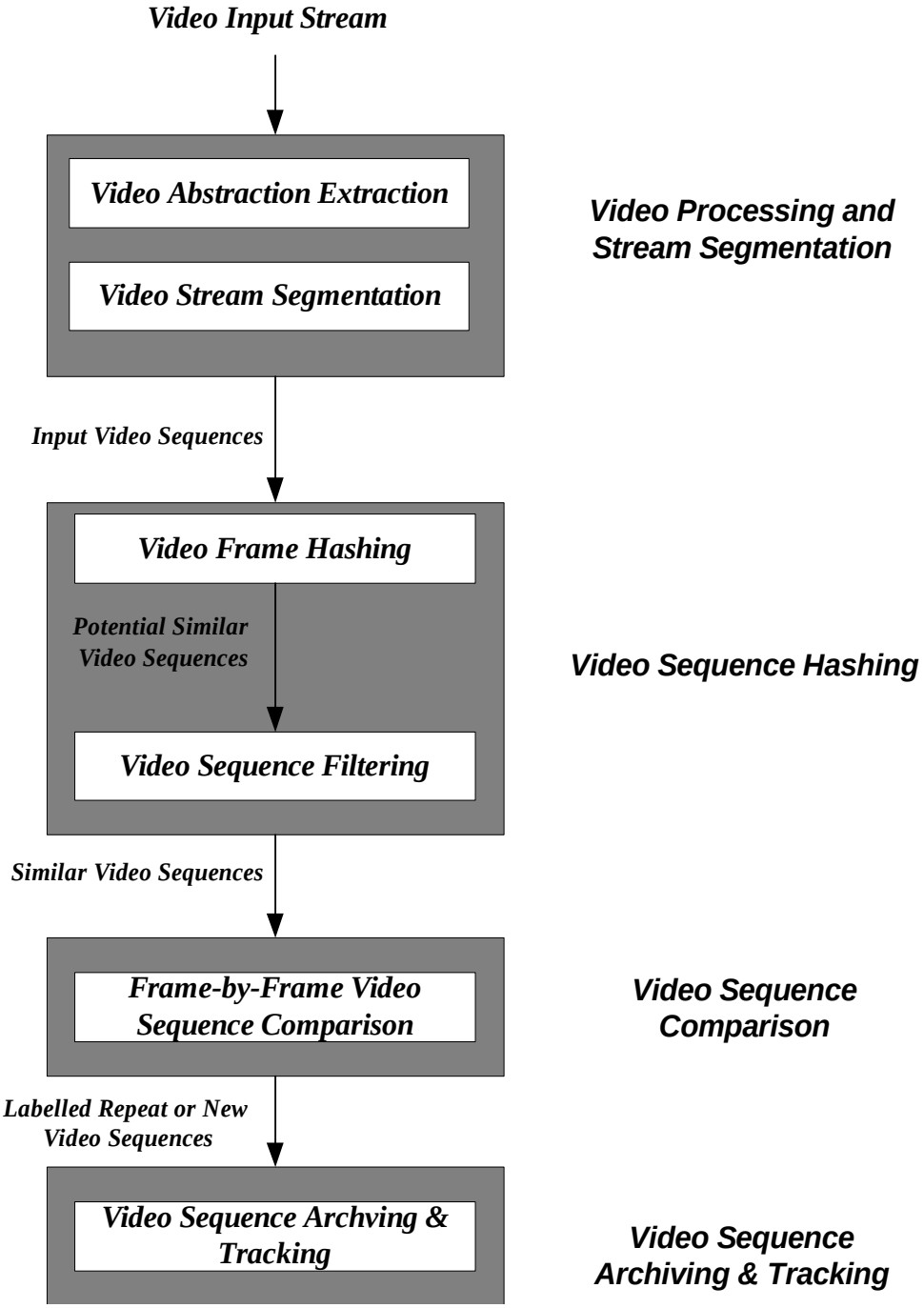


Figure 4.1 Block diagram of the Video Sequence Identification and Tracking System

4.1.1 Video Processing and Stream Segmentation Process

Our primary goal is to develop real-time algorithms to compare the current video sequence to a large archive of previously viewed video sequences. Since we do not want to overlook video sequences that are longer, more complete, versions of earlier sequences, we have chosen to compare abstractions for every image in the video sequence rather than to develop an algorithm based on comparing sequence features such as key frames. Since so many frames are to compared, the efficiency of the sequence abstraction algorithm is of prime importance. Because the color moment feature method [12] is compact, efficient, and accurate, we have adopted it as our video frame abstraction. The color moments used are the mean, the standard deviation and the skew of the Red, Blue and Green color components of the pixels in each video frame. Thus, regardless of the size of the video frame, each frame in the video sequence is represented by nine color moment values. The video broadcast time, or start and end broadcasting time and duration for each video sequence, is also stored. This temporal information about video sequences can be used as indices to track occurrences for each video sequence processed.

In our research, we define a video sequence as a single video shot. A video shot is an image sequence that represents continuous action and corresponds to a single action of the camera. The video stream is separated into individual video sequences using the video segmentation technique developed by the VISION project [15]. It is a feature-based algorithm that uses color histogram, image differences, and

average brightness to detect shot boundaries. It is able to recognize shot boundaries in real-time with 94% accuracy [13].

4.1.2 Video Sequence Hashing Process

For an archive of documentary programs, individual video sequences average 60 seconds in length. Thus, approximately 1,440 video sequences will be created daily. In order to determine if a video sequence is new or a previously broadcast sequence, a total of 1,036,080 ((1,440-1)*(1,440/2)) video sequence comparisons are needed. Other video sources, for example news, may have shorter sequences and thus required more comparisons. Hence, it is important to do this video comparison efficiently.

In order to identify new video sequences in real-time, we must be able to compare the current video sequences to all archived video sequences efficiently. We have designed a video comparison algorithm based on hashing. The video hashing process consists of two major components, namely video frame hashing and similar video sequence filtering. We use the video frame hashing component to identify video sequences in the video archive that are potentially similar to the input video sequence. We then use the video sequence filtering component to determine if these potentially similar video sequences are truly similar to the input video sequence.

During video frame hashing, the nine color moments calculated for the current video frame are mapped from floating point numbers to integers to remove noise and group similar video frames together. These nine integer numbers are then concatenated to create a fixed-length color moment string. The color moment string

is used as the key for this video frame and also the hash value for this frame. The color moment string of video frame, along with its video sequence identifier and size, is stored in the hash bucket identified by the hash value. Thus, the color moment string and sequence identifier for every frame from the digitized video stream is stored in the hash table. All frames that share identical color moment strings are grouped together in the same hash bucket. All video sequences that have at least one frame in the same hash bucket as the input video sequence are considered potential similar video sequences.

While the video frame hashing process will detect similar video frames and identify potential similar video sequences, we require the second component of the video hashing process in order to identify truly similar video sequences. Potential similar video sequences are filtered to remove those sequences whose degree of similarity is below some threshold. Video sequence similarity is based on the lengths of the potential similar and input sequences as well as the percentage of frames in the two sequences that have identical color moment strings.

Based on informal observation, we consider two video sequences to be dissimilar if their size difference is greater than 10%. For videos of roughly the same size, we concluded experiments to establish a overlap threshold value to be used as the ceiling for potential similar video sequence filtering (see section 6.1.4). The overlap threshold is defined in term of the percentage of frames in the two video sequences that have the same color moment string. Video sequences that fail to

exceed the overlap threshold for any previously archived video sequence are deemed to be new video sequences, the first occurrence in the video archive.

4.1.3 Video Sequence Comparison Process

The method described above might result in false positive matches. There are two main sources of error. First, the color moment strings used for comparison are built from approximate color moment values. Second, the video sequence filtering technique considers only the percentage of similar video frames, ignoring their temporal ordering. Thus, the final step of our algorithm performs a more accurate, frame-by-frame, video sequence comparison. In the frame-by-frame video sequence comparison process, the absolute moment differences between video frames from the input video sequence and the similar video sequences are calculated. The original floating point color moment values are used and the differences are aggregated over the entire sequence. The absolute moment difference value calculated for each similar video sequence is compared with a moment difference threshold to determine whether or not the video abstractions are similar enough for the input video sequence to be considered a repeated sequence.

4.1.4 Video Sequence Archiving and Tracking

The goals of the video sequence archiving and tracking process are to: 1) record meta-information for each video sequence processed; 2) record the video

identification process results, i.e., the total number of new and repeat video sequences processed and whether or not each input video sequence is new or a repeat; and 3) record and control the total working size of the video archive captured by the video identification process.

In conjunction with the video sequence identification process, meta-information for a video sequence is recorded and kept in a video sequence index table data structure. The meta-information includes the video sequence identifier assigned, the captured time of video sequence, the length of the video sequence (number of frames extracted during the segmentation step), and also a video sequence identification label assigned during the matching process. The video sequence identification flag indicates whether the sequence is new or a repeat.

In our research, we designed a video sequence identification and tracking system that can process and store an archive of at least 24 hours video worth of continuous video broadcast in real time. Continuous processing in excess of 24 hours is handled by using a sliding window wherein only the most recent 24 hours of video are kept. Using the information stored in the index table, one can track video sequences broadcast at any time within the defined 24 hours of time window, and hence easily reproduce the whole video stream or find all repeats for a given sequence.

4.2 System Architecture Design and Definition

As described in Section 4.1, there are four main processes in our Video Sequence Identification technique and Tracking System: 1) Video Processing and Stream Segmentation; 2) Video Sequence Hashing; 3) Video Sequence Comparison; and 4) Video Sequence Archiving and Tracking. We designed and developed two stand-alone systems with a total of four main software component modules (Figure 4.2) to support the four processes of the video identification and tracking system.

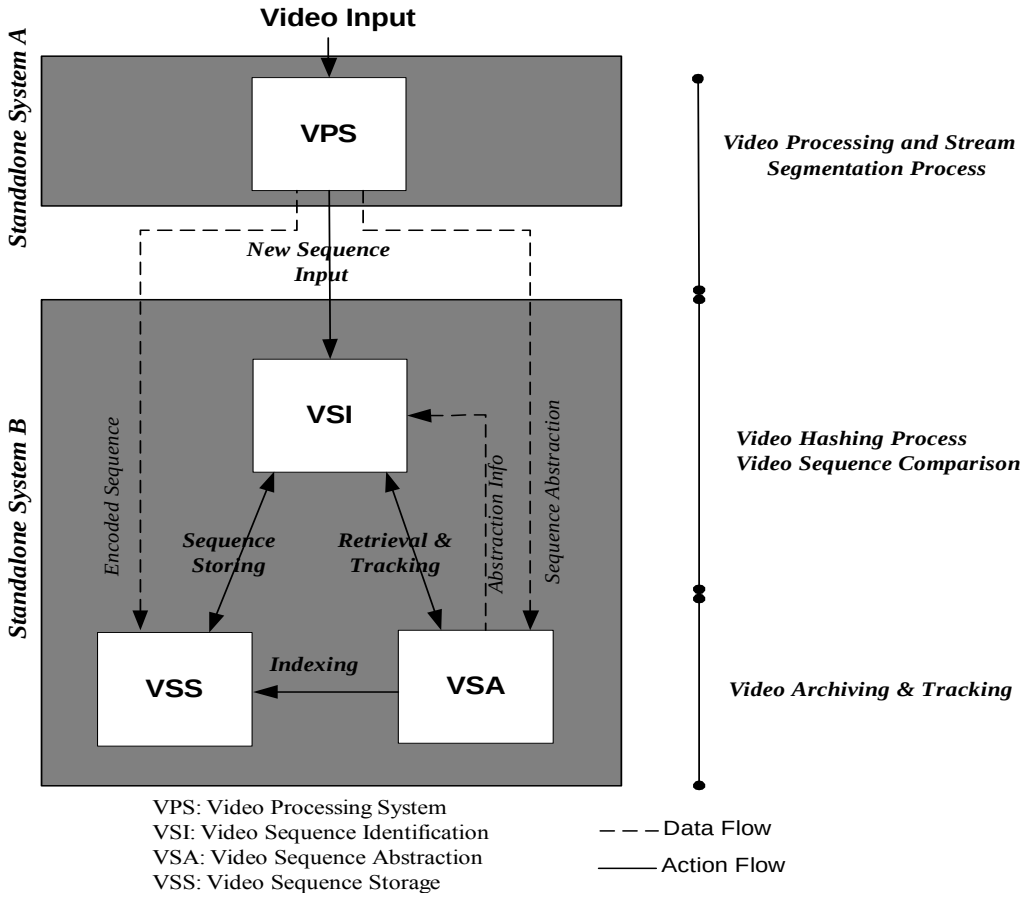


Figure 4.2 System flow diagram of the Video Sequence Identification and Tracking System

These four main components are: 1) Video Processing System (VPS); 2) Video Sequence Identification module (VSI); 3) Video Sequence Abstraction Module (VSA); and 4) Video Sequence Storage (VSS). The VPS component is one stand-alone system and the other three components are modules of the second stand-alone system. As shown in figure 4.2, video input is first digitized and segmented into shots using the Video Processing System (VPS). The newly digitized video sequence is then compared to all previously archived video sequences using the Video Sequence Identification (VSI) module. If no matches are found, the video sequence is deemed to be a new video sequence; i.e., the first occurrence of a video sequence. The video sequence, together with its video abstractions, are sent to the Video Sequence Abstraction (VSA) and the Video Sequence Storage (VSS) modules for analysis and storage.

a. Video Processing System (VPS)

The main functions of the VPS are video signal digitization, video abstraction extraction, and video shot detection and segmentation. The VPS receives a video source, digitizes the video signal into individual images in RGB color format, and segments the video stream into different shots. A video abstraction based on color moments is extracted from each frame in the shot. The color moments and shot boundaries are passed along to the Video Sequence Identification module (VSI). The digitized video frames are encoded into RealMedia format and sent to the Video Sequence Storage (VSS) for storage.

b. Video Sequence Identification (VSI)

The main focus of this research is the design of video sequence identification technique used by the Video Sequence Identification Module. To compare a new shot's video abstraction to a potentially large collection of stored video abstractions efficiently, a technique combining hashing and filtering was designed.

c. Video Sequence Abstraction Module (VSA)

The video abstraction information for each video sequence is kept in the Video Sequence Abstraction (VSA) Module. This video abstraction consists of the nine color moment values of each video frame stored in color moment string format, and meta-information such as the video sequence size and the temporal information of video sequence. We also keep the raw nine color moment values of each frame that are needed in the video sequence comparison process. The VSA module consists of a video sequence index table, which captures the meta-information and results of the identification process of each video sequence and the file index information to its video abstraction and video clip.

d. Video Sequence Storage (VSS)

The Video Clip Storage (VCS) stores unique video clips encoded in RealMedia and also their video abstraction (nine moment values per frame) in text format.

The standalone Video Processing System (VPS) used in this research is a modified version of the Video Processing Software Module designed and implemented in the VISION project [15] for video shot detection and segmentation. It was implemented in *C* with the *OSPREY* digitizer board on a Windows NT platform. The second standalone system (*standalone system B*) containing the VSI, VSA, and VSS modules, was also implemented using the *C* language on a Windows NT platform. The main reasons for dividing the video segmentation and abstraction process and video identification and tracking process into two separate standalone systems are: 1) to provide system modularity and independence for future expansion and enhancement of both systems; and 2) to interact easily with the VPS. Output from the VPS is saved in disk files in a common folder that is accessible by multiple systems for data access and sharing. The following pseudocode describes the overall system flow of the video identification and tracking process :

*/** The Overall Process Flow of Video Sequence Identification and Tracking System **/*

System Inputs :

1) an existing video hash table and video index table in text files

2) system default parameters in a text file

3) video sequences created by the VPS module that are stored in a common input folder accessible by both VPS standalone system and the identification system.

/ **System Restart** */*

system initialization and parameter reading;

/ **Main video identification process infinite loop***/*

for (; ;) {

if (there is input video sequence found in input folder) {

/ **Input Video Sequence reading** */*

Read original moment values of each frame of the sequence;

Create Color Moment String for each frame;

/ Video Sequence Identification (VSI) Module – **Phase II** */*

Do video sequence hashing process;

Do video sequence comparison;

/ Video Sequence Abstraction (VSA) & Video Sequence Storage (VSS) – **Phase III***/*

Record result and update video index table – remove any expired sequences;

Update video hashing table – remove any expired video frames;

}

/ wait 5 seconds and then go back to input folder to check for any new input sequence */*

Wait 5 seconds;

Go back to the input folder and check for any new input video sequence;

}

System Outputs:

1) Result log files

2) System standard log files

3) Video hash table and video index table in text files

– they are updated automatically in every user-defined interval during the continuously running process

Chapter 5

Design of Video Sequence Identification and Tracking

5.1 Video Processing and Segmentation Process

The video processing and segmentation process involves the abstraction and creation of video sequences. Video data streams need to be processed and segmented into video sequences before the process of video sequence identification. We first extract video abstractions and segment the video stream into multiple video sequences. We then process video sequences to identify new video sequences and record temporal information that can be used to track video sequence occurrences or to reproduce the original video stream. This section describes our methods of video sequence abstraction and video segmentation in detail.

5.1.1 Video Frame Abstraction

We represent video sequences using image features extracted from each video frame for video processing and segmentation. A wide variety of possible features and feature abstraction techniques have been used by the research community. The following sub-sections describe abstracting video frames using properties of color, texture, and shape. We conclude this section with a discussion of why, among the various possibilities, we chose to use color moments as our video abstraction.

5.1.1.1 Color Features

Image abstraction based on color features has been studied extensively and there are multiple ways of representing an image using color features. A few of the most commonly used color measures are the color histogram, dominant colors, and statistical color moments. The following subsections discuss each of these color measures.

Color Histogram A color histogram is a function showing, for each color *c*, the number of pixels in the image that have this color. In other words, it describes the distribution of colors in an image or video frame. Let

$$F(x,y), = c , \quad x \in [0..M-1] , y \in [0..P-1] \qquad \text{(Equation 5.1)}$$

where M is the total number of row,

P is the total number of column,

c is a color value.

be an image consisting of N colors. Then the normalized histogram is given as:

$$h(c) = \frac{1}{MP}_{xy} \begin{Bmatrix} 1 & if & F(x,y) = c \\ 0 & & otherwise \end{Bmatrix}$$

(Equation 5.2)

where $M*P$ is the total image size.

The color histogram is invariant to translation, rotation, change of angle of view, change in scale, and small occlusions. The similarity between two images can be computed using the following simple distance measurement:

$$D(A,B) = \sum_{c=1}^{N} | h_A(c) - h_B(c) |$$

(Equation 5.3)

A color histogram only records an image's overall color composition, so images with very different appearances can have similar color histograms.

Dominant Colors Only a small number of colors are used to represent an image. Swain and Ballard [38] have shown that using only a few dominant colors for image comparison did not lower the performance. It could even enhance it by getting rid of noise often represented by irrelevant colors.

Statistical Color Moments Another way of representing an image is the use of its first three color moments, namely the mean, standard deviation and skew of each primary

color component found in the RGB color space (see Appendix C). The mean of a set of values is used to estimate the value around which central clustering occurs. The standard deviation describes the "width" or "variability" around the mean value. The skew characterizes the degree of asymmetry of a distribution around its mean. Skew is non-dimensional and characterizes only the shape of the distribution. Higher moments, involve more manipulations on the input data, are almost always less robust than lower moments that involve only linear sums or, the lowest moment of all, counting. Therefore, higher moments such as fourth moment or above are rarely used to represent the content of an image.

Given $I \in [R, G, B]$, the first three moment values are defined by the following equations:

$$Mean = U_i = \frac{1}{MP} \sum_j G_{i,j}$$

(Equation 5.4)

$$S.Deviation. = S_i = \left[\frac{1}{MP} \sum_j (G_{i,j} - U_i)^2\right]^{1/2}$$

(Equation 5.5)

$$Skew = S_i = \left[\frac{1}{MP}\sum_j (G_{i,j} - U_i)^3\right]^{1/3}$$

(Equation 5.6)

where $G_{i,j}$ is the value of the *i*-th color component of the *j*-th image pixel,

$M*P$ is the total image size.

The distance between two color distributions representing two images (image A and image B) is given as:

$$D(A,B) = \sum_{i=R,G,B} (|U_i(A) - U_i(B)| + |V_i(A) - V_i(B)| + |S_i(A) - S_i(B)|)$$

(Equation 5.7)

Even with only nine moment values per image, studies in [12] and [43] have found that this method of representing images is more robust than color histogram methods.

5.1.1.2 Texture Features

Texture is observed in the structural patterns of surfaces of objects such as wood, grain, sand, grass, and cloth. The term texture generally refers to repetition of basic texture elements called *texels*. A texel contains several pixels whose placement could be periodic, quasi-periodic or random. Figure 5.1 lists several texture measures. In image analysis, texture is broadly classified into two main categories, statistical and structural. The mosaic model is based on the combination of the statistical and the structural approaches.

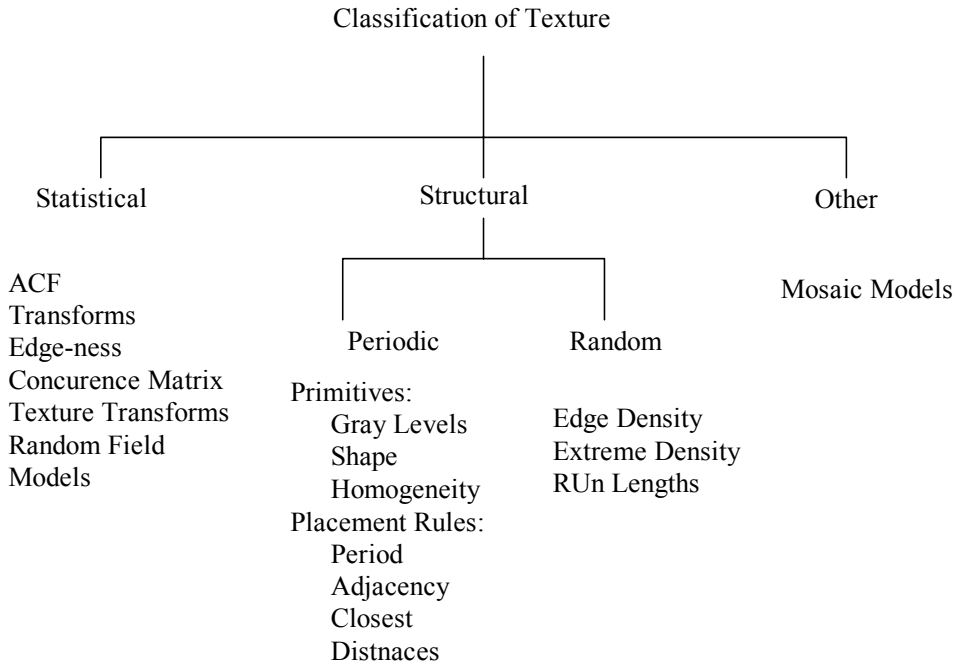


Figure 5.1 Image texture categorization model

Statistical Approaches Textures that are random in nature, for example, the realizations of random fields such as wood grain and sand, are well suited for statistical characterization. Three common statistical models used for measuring texture of an image are: 1) the autocorrelation function (ACF) to measure coarseness of texture; 2) image transforms to estimate coarseness, fineness, and orientation of texture; and 3) histogram features to measure coarseness and the orientation-independent spread of the texture. Appendix A presents the mathematical description of each model. The other statistical texture methods include the use of edge density to measure the coarseness of the random texture in which the edge density is measured by the average number of edge pixels per unit area.

Structural Approaches Purely structural textures are deterministic texels, which repeat according to some placement rules. A texel is isolated by identifying a group of pixels having certain invariant properties that repeat in the given image. The texel may be defined by its gray level, shape, or homogeneity of some local property such as size, orientation, or second-order histogram. The placement rules define the spatial relationships between the texels and these spatial relationships may be expressed in terms of adjacency, closest distance, and periodicities. The texture is labeled as being *strong* in the case of deterministic placement rules. In case of the randomly placed texels, the associated texture is called *weak* and the placement rules may be expressed in terms of measures such as edge density, run lengths of maximally connected texels, and relative extrema density (the number of pixels per unit area showing gray levels that are locally maxima or minima).

Mosaic Model The mosaic model, which combines statistical and structural approaches, is used to represent random geometrical processes. A mosaic model could define rules for partitioning a plane into different cells, where each cell contains a geometric figure whose features have prescribed probability distributions.

5.1.1.3 Shape Features

The shape of an object refers to its profile and physical structure. These characteristics can be represented by the boundary, region, moment, and structural representations. These representations can be used for matching shapes, recognizing

objects, or for making measurements of shapes. Figure 5.2 lists several useful features of shape.

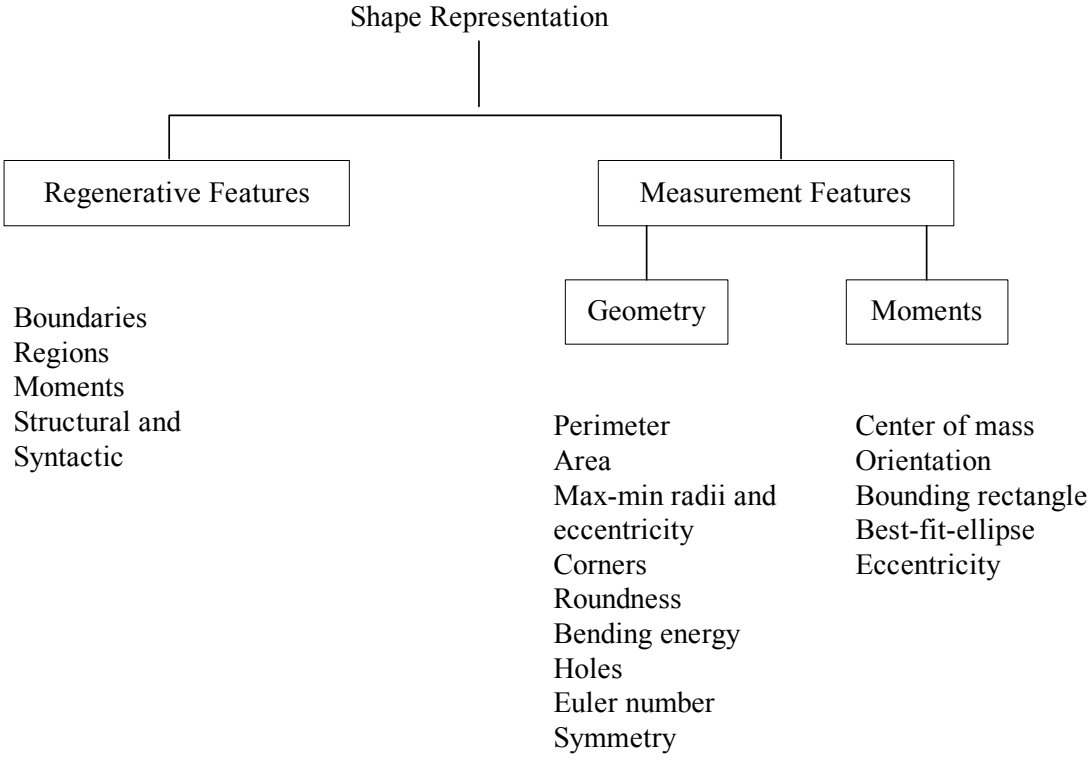


Figure 5.2 Shape features

Many shape features can be conveniently represented in terms of moments (Figure 5.2). One of the common shape measurements using moment values is the moment invariants. Moment invariants are spatial properties of connected region in images that are invariant to translation, rotation and scale. They are useful because they define a simple calculated set of region properties that can be used to perform image similarity queries using the Euclidean distance given as:

$$D_{mom}(A,B) = \sqrt{\sum_{i=1}^{7} \left(W_i^2 (M_i(A) - M_i(B))^2 \right)} \qquad \text{(Equation 5.8)}$$

where $M_i()$, $i=1….7$, is the first seven moment invariants

(see Appendix B for the Moment Invariant Measurement),

A and B are the two comparing images,

W_i is a scale factor calculated such that all moments of all images are normalized into [0…1] range.

5.1.2 Video Sequence Abstraction Using Color Moments

Our primary goal is to develop real-time algorithms to compare an input video sequence to a large archive of previously viewed video sequences. Since we do not want to overlook video sequences that are longer, more complete versions of earlier sequences, we have chosen to compare abstractions for every frame in the video sequence rather than to develop an algorithm based on comparing key-frames only.

Since so many frames are to compared, our video abstraction must be able to represent the content of each frame compactly while still preserving the individuality of the frames as much as possible. We eliminated using either texture-based or shape-based frame representations as our frame abstraction because they are input domain dependent and only work well on a limited range of video sources. On the other hand, a color-based abstraction, such as statistical moments, provides a good description of the overall color characteristics of a frame and has been shown to work well for video sources in general. We decided to use a set of nine color moments to

represent the content of each video frame captured. These nine color moments are the first three color moments of the Red, Blue and Green primary color component for each video frame, namely the mean, the standard deviation and the skew. Equations 5.4, 5.5 and 5.6 presented in the previous section show the mathematical calculation of these values.

We chose to use color moments [11] to represent video frames because they are compact, efficient to calculate, effective for similarity computation and require little storage, regardless of the size of the video frame. This video abstraction will be used as the data input for our video sequence identification algorithm. The start and end broadcasting time and duration for each video sequence are also stored. This temporal information about video sequences can be used as indices to track occurrences for each video sequence processed.

5.1.3 Creation of Video Sequences

In order to identify and track video sequences, we first must define the meaning of a video sequence. In this paper, a video sequence is defined as a single video shot. A video shot is an image sequence that represents continuous action. It corresponds to a single action of the camera. The video stream is segmented into multiple shots using the content-based video segmentation technique developed in VISION project [15].

When making a video for broadcast, three major steps are involved: 1) shot creation; 2) editing; and 3) final cut assembly. Editing decides the ordering of the shots and the choice of transitions between two consecutive shots, creating an edit

decision list. The final process assembles the shots and transition effects into the final cut. Figure 5.3 illustrates a sample of the video production model.

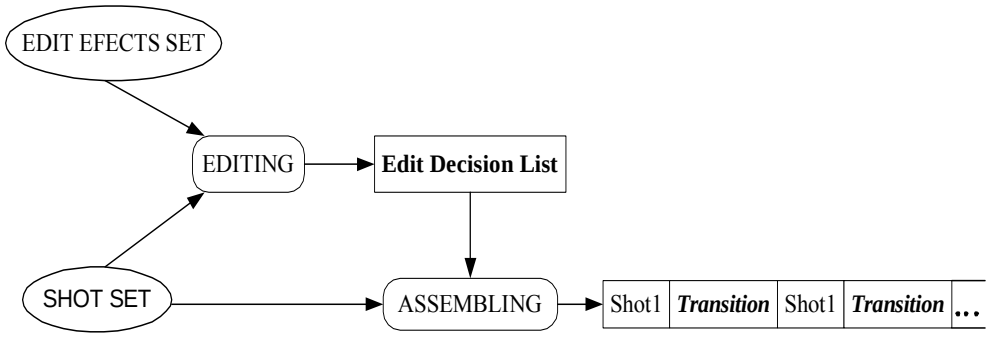


Figure 5.3 The video product model

The creation of video shots from a video stream is in fact the reverse process of assembling and detecting the transitions created during editing. We identify the boundary between the shots as one frame in the middle of the transition. The VISION system uses color histograms (Equation 5.11), intensity differences (Equation 5.10) and average brightness (Equation 5.9) to detect the transitions between video shots. The following three measurements had been defined in [4] for video shot detection:

■ Average Brightness (B):

$$B(t) = \sum_{xy} I(x, y, t)$$

(Equation 5.9)

where t is the time coordinate *(*frame = *t)* of a video sequence,
x is the horizontal coordinate of the frame,

y is the vertical coordinate of the frame,

I is the pixel value.

■ Intensity Difference (dP):

$$dP(t,dt) = \sum_{xy} | I(x,y,t) - I(x,y,t-dt) |$$

(Equation 5.10)

where *t* is time coordinate (frame = t) of the video sequence,

dt is the total time difference (number of frames),

I is the pixel value,

x & *y* are the horizontal and vertical coordinates of the frame.

■ Histogram Difference (dC) :

$$dC(t,dt) = \sum_{c} | h(c,t) - h(c,t-dt) |$$

(Equation 5.11)

where

$$h(c,t) = \sum_{xy} \begin{cases} 1 & if \quad I(x,y,t) = c \\ 0 & otherwise \end{cases}$$

(Equation 5.12)

I is the pixel value,

t is the time coordinate (tth frame) of the video sequence,

x& y are the horizontal and vertical coordinates of the frame,

c is one of the defined pixel values

Transitions may be fast and sharp, called a cut, or relatively long and gradual, called a smooth edit. If we assume the transition frame does not belong to the shots, there should be a large difference in pixel intensity and histogram between the last frame of shot *n* and the first frame of shot *n+1*. Therefore, if we look at the differences between two consecutive frames, then we should detect the cuts and hence boundaries of the video shots. If we look at the differences between two frames *dt* distant from each other, we should be able to detect smoother transitions, assuming our choice for *dt* is correct. Usually, when there is a lot of motion in a picture, the value *dP(t,1)* (pixel intensity difference) is high. Moreover, smooth edits are applied on still images, so the *dP(t,1)* must be very small when we have a smooth edit. In conclusion, in order to detect video shot boundaries with smooth edits, we need to look at values of *dC(t,dt)* and *dP(t,dt),* where *dt >1*, but only if the values of *dC(t,1)* and *dP(t,1)* are small enough.

The experimental results of VISION project in [13][15] suggested the use of three values of *dt*: 1, 5, 10 to produce satisfactory video shot detection results. Three threshold values, namely B_{thresh}, C_{thresh}, and P_{thresh} were defined for the video shot detection process. There is a shot change if the following statement is true:

$B(t) < B_{thresh}$

(The brightness is too low, indicating a blank transition frame)

OR

$((dC(t,1) > C_{thresh})$ AND $(dP(t,1) > P_{thresh}))$

(Both histogram difference and intensity difference are higher than a given threshold values)

OR

$(((\ dC(t,1) < C_{thresh}\ )$ AND $(\ dP(t,1) < P_{thresh}\ ))$ AND

$((\ dC(t,5) > C_{thresh}\ )$ AND $(\ dP(t,5) > P_{thresh}\ )))$

(Both histogram difference and intensity difference are below than a given threshold values, and the both histogram difference and intensity difference measured a distance of 5 frame away ($dt=5$) are higher than a given thresholds)

OR

$(((\ dC(t,1) < C_{thresh}\ )$ AND $(\ dP(t,1) < P_{thresh}\ ))$ AND

$((\ dC(t,5) < C_{thresh}\ )$ AND $(\ dP(t,5) < P_{thresh}\ ))$ AND

$((\ dC(t,10) < C_{thresh}\ )$ AND $(\ dP(t,10) > P_{thresh}\ )))$

(Both histogram difference and intensity difference measured for $dt=0$ and $dt=5$ are lower than a given threshold, but are higher than the same given threshold for measurement at $dt=10$)

The values chosen for each threshold were experimentally selected [4].

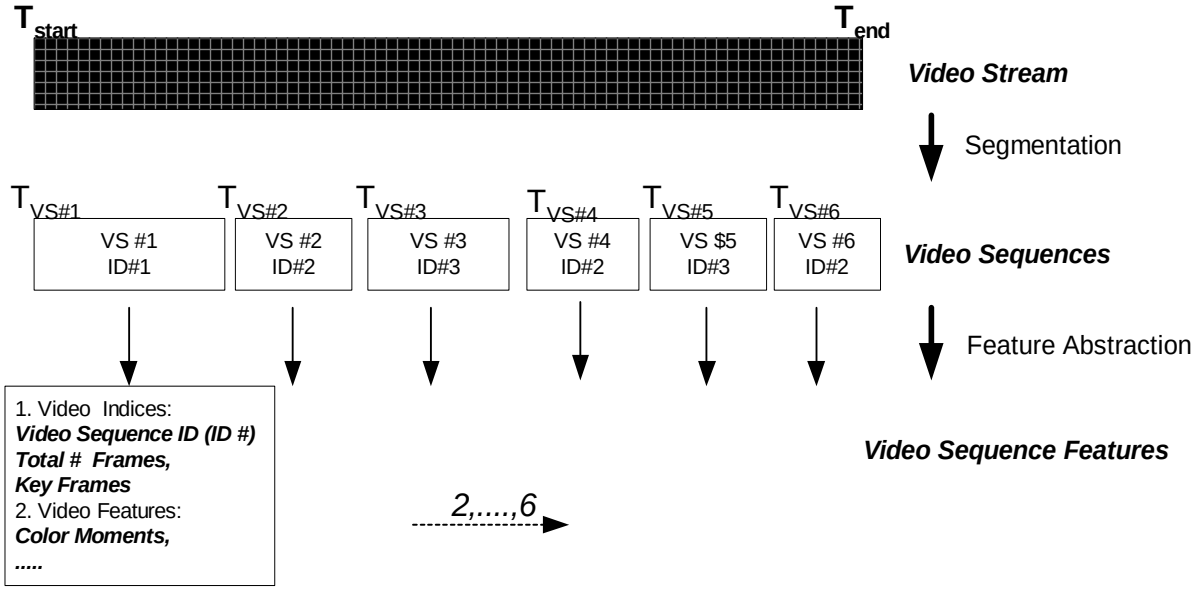


Figure 5.4 Video segmentation and abstraction extraction

Figure 5.4 illustrates the video stream segmentation and processing performed by the Video Processing System (VPS). The average brightness, pixel intensity and color histogram of each frame from video stream is extracted and is used to determine shot boundaries during the video sequence creation process. Each created video sequence is uniquely named using the date and time of its creation. Other information, such as size of each video sequence, representative key frames, and the first three color moments are also extracted during the video processing and segmentation process.

This dissertation will not go into depth into the design and implementation of the Video Processing System (VPS). For details of the design and implementation technique, please refer to Sylvain Bouix's Master's Thesis [4] and Gauch [13].

5.2 Video Sequence Hashing Process

Once we have a newly captured video sequence, we send its abstraction to the Video Sequence Identification and Tracking Components (VSI and VSS) to determine if this is a replay of a previously seen sequence or the first occurrence of a new sequence. The first step of the video sequence identification technique is the video sequence hashing process, which is designed to identify video sequences in the archive that are similar to the input video sequence. Figure 5.5 illustrates the conceptual flow of the video hashing process. Referring to the figure, the video sequence hashing technique is divided into two main components: 1) video frame hashing to detect potential similar video sequences; and 2) video sequence filtering to identify similar video sequences. The following sections describe the design and implementation of each component.

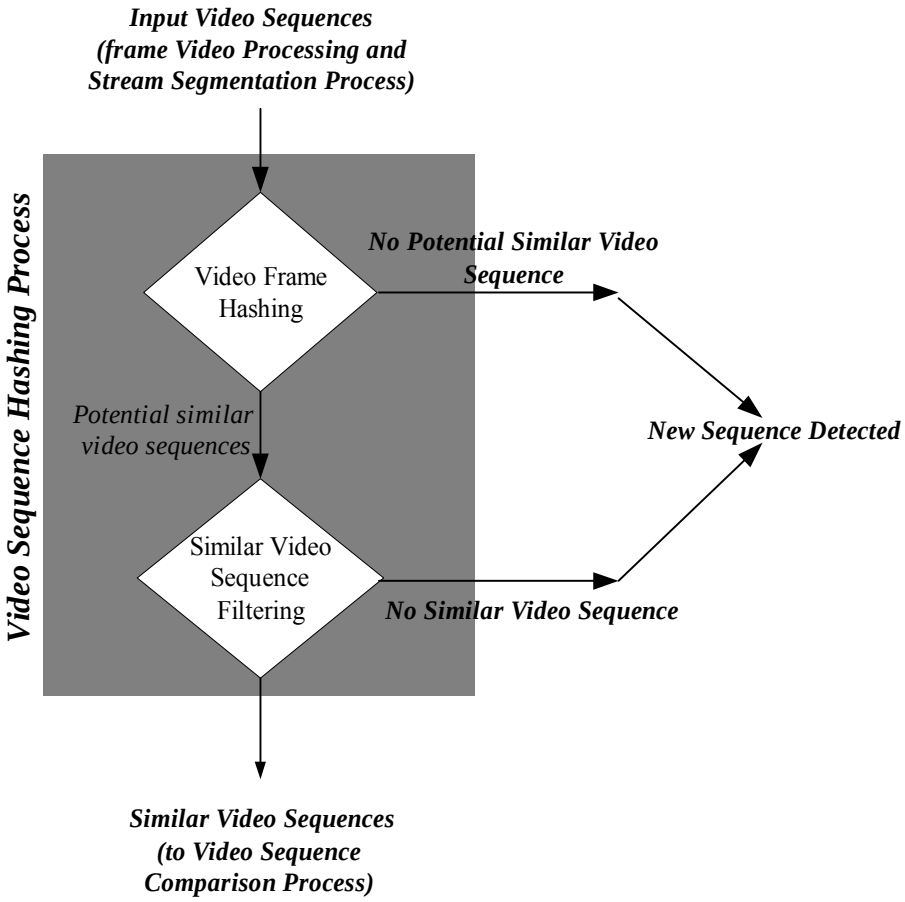


Figure 5.5 Block diagram of Video Hashing Process

5.2.1 Video Frame Hashing

During video frame hashing, the nine color moments for each frame extracted in VPS are mapped from floating point values to integers and then concatenated to create a fixed length color moment string. These color moment string is used as a hash key for the frame. The frame's color moment string, video sequence identifier, and video sequence length are stored in the appropriate hash table bucket. This process groups all video frames with similar color moment strings into the same bucket of the Video

Sequence Abstraction Module. Figure 5.6 illustrates the flow diagram of the video frame hashing. Referring to the flow diagram of the figure, the first step of the process for each new input video sequence is to generate a hash key for each video frame. Then, we perform video frame hashing for each hash key to collect potential similar video sequences, defined as those having at least a single frame with the same color moment string as a frame in the input video sequence.

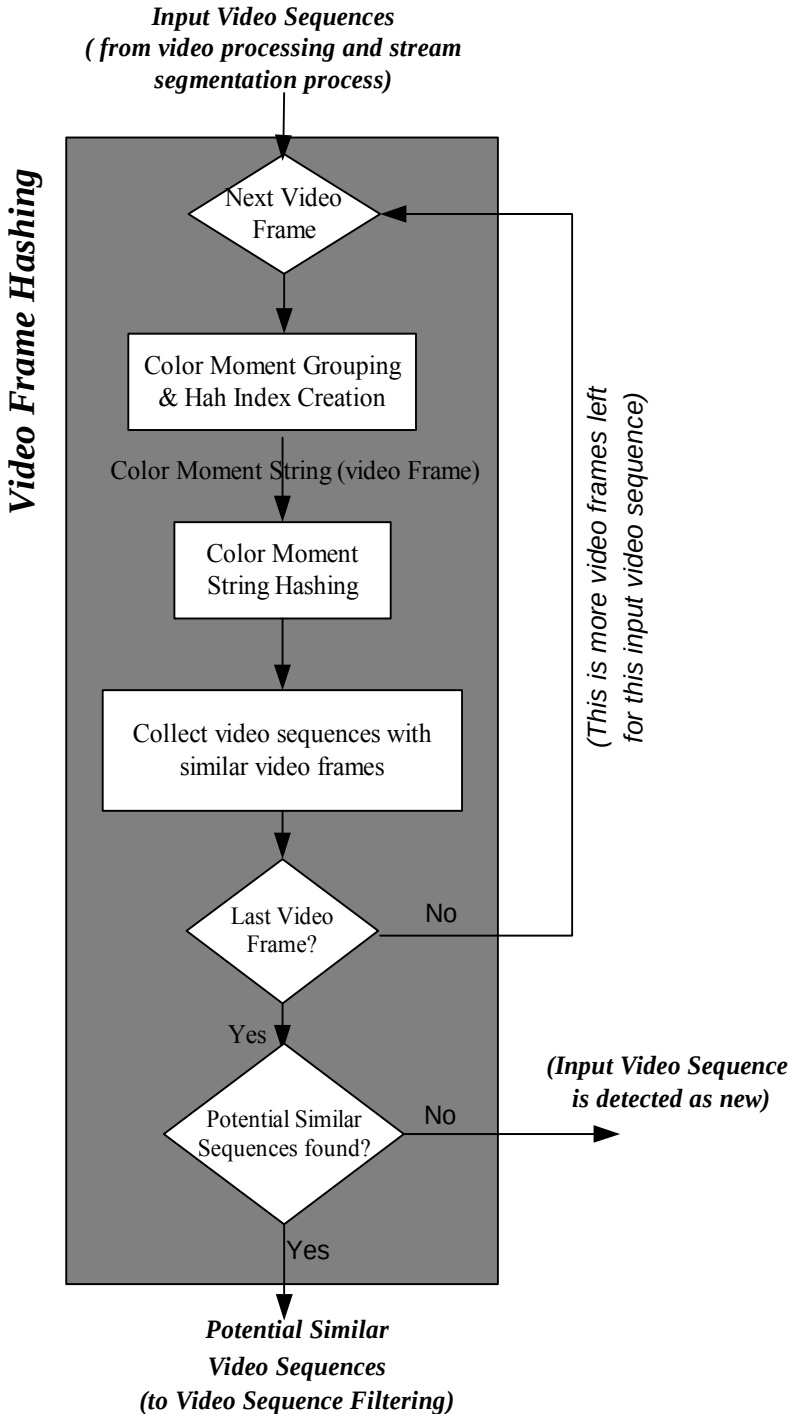


Figure 5.6 The flow diagram of Video Frame Hashing

5.2.1.1 Video Hash Table Design

Figure 5.7 illustrates the video hash table data structure designed for the video hashing process. Each hash table bucket contains a frame's color moment string, video sequence identifier, a clip counter that captures the total number of video sequences containing this video frame, and a pointer that points to a sorted linked list. The linked list captures the meta-information for each video sequence, namely: 1) the video sequence identifier (ClipID); 2) the temporal order (frame position) of the index video frame in this video sequence (FramePos); 3) the size of the video sequence (Size); 4) the date when the video sequence was created (Time); and 5) a pointer that points to the next linked list element. The linked list for each hash bucket is sorted by the video sequence identifier.

For our experiments, we created a hash table large enough to store 24 hours of video frames. A sliding window mechanism was used to control the maximum number of video sequences stored in the hash table at one time. Assuming a 15 frame/second video stream, 24 hours of video will contain 24 hours * 3600 seconds/hour * 15 frames/second = 1.296 million video frames. The maximum numbers of buckets required to handle the worst case (when all 24 hours worth of video frames are unique) will be 1,296,000. In our work, we implemented a memory-based hash table with a bucket size equals to 3,600,000 to handle a total of 24 hours of video window hashing size. This will require nearly 110MB of memory to perform the memory-based hashing. The following shows how the memory size requirement for the memory-based hashing is calculated:

(refer to Figure 5.7 for hash table structure description)

Memory size assigned for the Color Moment String field = 54 bytes

Memory size assigned for the Total Number of Clips field = 4 bytes

Total memory size assigned for other fields in each hash bucket field = 8 bytes

Total memory size required for one hash bucket = 54+4+8 = 66 bytes

*Total memory size assigned for one linked list element = 5*4 = 20 bytes*

Therefore,

Total memory required for storing one video frame in a hash bucket

= Total memory size of one hash bucket + Total memory size required for one linked list = 66 + 20 = 88 bytes

*Total memory size needed in order to store a 24 hours worth of unique video frames = 88 bytes * 3.6 million = 317 MB*

For handling video hashing with a very large video window size, a disk-based hash file could be used.

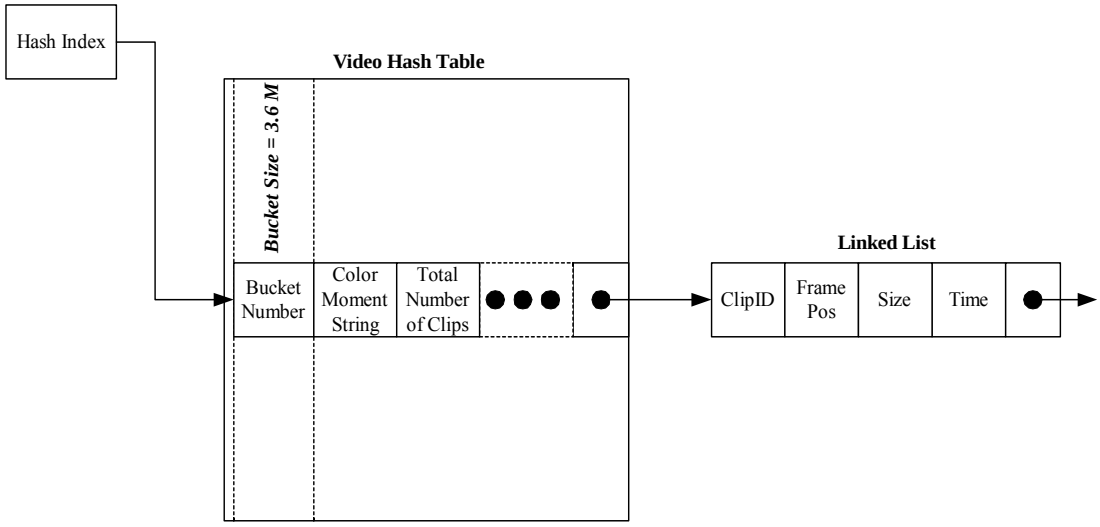


Figure 5.7 Hash table data structure for Video Frame Hashing Process

Assume that the video hashing table contains *n* hours of previously stored video sequences. A video sequence is captured and we want to whether or not this video sequence is a new or repeat video sequence. The following pseudocode describes steps performed during the video frame hashing process:

/*** ***Video Frame Hashing Process Pseudocode*** ***/

Component Input *: A linked list containing color moment strings for each video frame of the input video sequence*

/ **Component 1** - Video Frame Hashing */*

for (each color moment string of the input video sequence) {

Do hash index generation;

Do video table hashing to find similar video frames;

If there is a hit { // sequences with at least one similar video frame to the input video sequence

Traverse the matching linked list { // record result

1. record meta-information for each video sequence stored in each element of the linked list into an output linked list;

2. increment similar frame counter of a video sequence for each color moment string matching;

}

}

next color moment string

}

***Component Final Output**: A linked list containing video sequences having at least one video frame similar in the same bucket as a frame in the input video sequence*

5.2.1.2 Hash Index Generation

An important element of the video frame hashing process is the creation of hash index for each video frame of an input video sequence. The nine color moments calculated in the video processing system are concatenated to form the color moment string as shown below:

MNumber(n) = { M[Red(n)], M[Green(n)], M[Blue(n)]

S[Red(n)], S[Green(n)], S[Blue(n)]

Skew[Red(n)], Skew[Green(n)], Skew[Blue(n)] }

where

Mnumber(n) = The color moment string of a video frame *n,*

M[] = Mean value (0…256), (refer to Equation 5.4)

S[] = Standard Deviation (0…256), (refer to Equation 5.5)

Skew[] = Skew value (-256…256) (refer to Equation 5.6)

The color moment string is used as the hash index to determine the bucket in which the video frame is stored. A hashing experiment using a concatenation of all digits of the original float numbers of the nine moment values as the hashing key proved to be ineffective due to noise introduced by both transmission and digitization of the video source. A test of multiple passes of moment calculations for the same video frame showed that the moment values calculated for a single video frame can vary somewhat, making identical values unlikely for repeated video broadcasts. The test results also showed that the error in moment values calculated from a single video frame could range from –5.0 to +5.0. Therefore, we decided to ignore the least significant digit for each color moment. This results in a 10 to 1 mapping of raw moment values (see Table 6.1 for color moment mapping), e.g., the skew value will be mapped from (-256…256) to (-25…25). Figure 5.8 illustrates a sample mapping of the original nine color moments of a video frame into a color moment string.

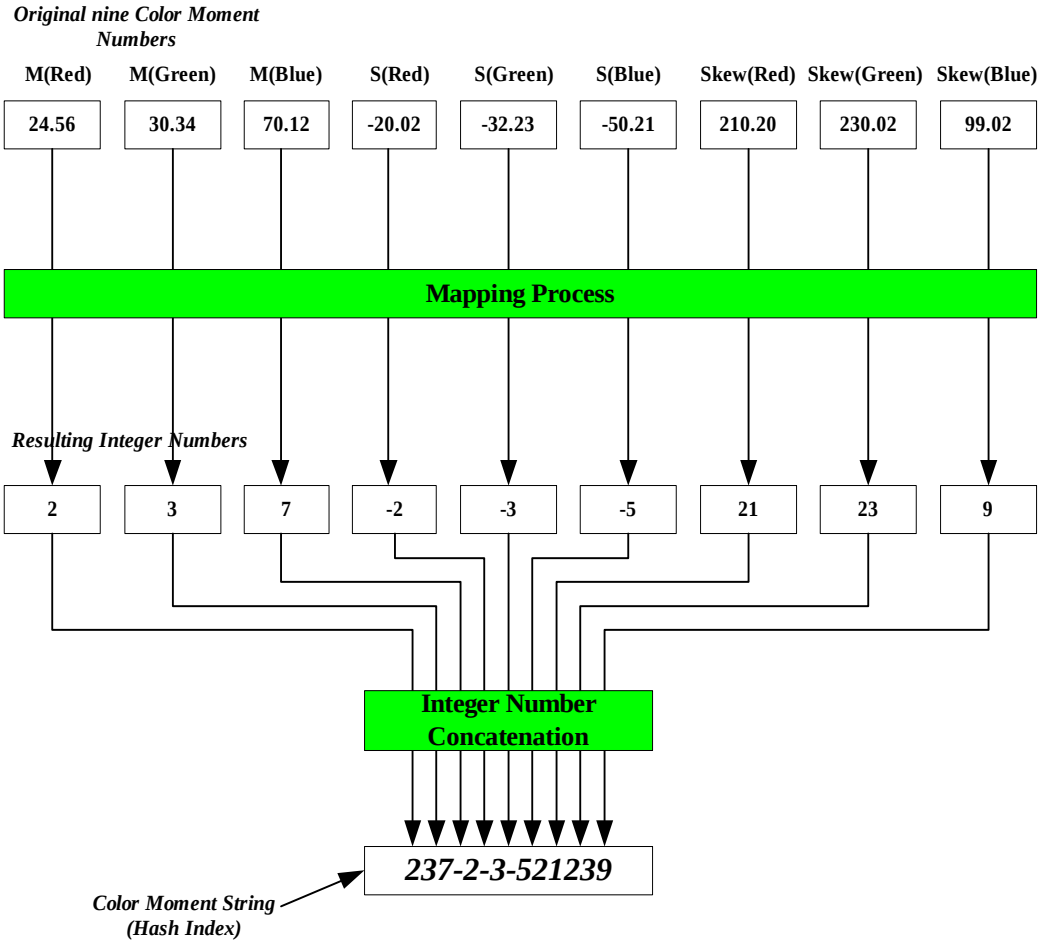


Figure 5.8 An example of color moment mapping process

The choice of the mapping ratio is important for the accuracy and speed of the video identification process. If the mapping ratio is too small, then highly similar video frames could fall into different buckets and hence actual repeated frames will be missed. For example, due to both transmission and quantization error, the nine color moment values for two identical video frames are calculated and are equal to 19.40 and 20.40 respectively. The absolute moment difference (1.0) between these two video frames indicates that they are indeed identical. Nevertheless, with a 10 to 1 mapping ratio, we will get different values (19 versus 20) for the 9 integer

components of their color moment strings and they will be hashed into different buckets. Therefore, identical video frames having moment values close the any one of the boundaries of the integer mapping numbers will tend to be mis-mapped into different video bucket even though their difference moment values are small enough to be considered as identical to each other.

A mapping ratio that is too large will result in dissimilar video frames ending up in the same bucket, requiring more work to be done (increase in total video comparison time) during video sequence comparison. However, a too large mapping ratio will not adversely affect the final accuracy of the sequence identification algorithm.

In order to select a mapping ratio that strikes a balance between speed and accuracy, we conducted an experiment to measure the mis-mapping error rate for 'identical' frames for various mapping ratios. We randomly selected a total of 25,916 video frames from six video sequences in the video archive. For the purpose of this experiment, two video frames were considered identical if the absolute moment difference of these two video frames was lower than a moment difference threshold of 10.0. A mis-mapping was recorded if the two video frames were identical but had different color moment strings. With 25,916 test video frames, over 335 million video frame pairs were compared. The mis-mapping error percentages for mapping ratios were recorded and the results are shown in Table 5.1. As expected, a smaller mapping ratio resulted in more similar video frames being mapped into different buckets and hence a higher mis-mapping error percentage. With a mapping ratio of

10 to 1, we recorded an error percentage of close to 1%, which was acceptable for our video frame hashing technique.

	Color Moment String Mapping Ratio			
Moment Difference Threshold	**5**	**10**	**15**	**20**
10.0	1.19%	*1.12%*	0.98%	0.90%

Table 5.1 Error percentage of color moment string mis-mapping

The total possible number of color moment strings can be calculated as follows:

Number of color moment strings = $R^M = 51^9 = 2.33 \times 10^{15}$

where,

R = Total possible integer value of each moment after mapping (-25…25)

M = Total number of color moments

The generation of the hash bucket index value using the hashing key is described by the following pseudo code:

```
/* Hash Bucket Index Generation */
Index =0;
Sum =0;
TableSize = Size of Hash Table = 3.6M;
ColorString[M] = Color Moment String;
For (x=0; x < strlen(ColorString); x++)
```

*Sum = (Sum*19) + ColorString[x]*

Index = Sum% TableSize

Since the number of possible hash keys is many orders of magnitude larger than the number of hash buckets (3,600,000), we ran a simulation to evaluate the distribution of hash keys produced by our hash function. The simulation showed that the hash keys produced from input video frames produced an acceptable collision rate of 2 to 1. The simulation results validated our choice of using color moment string as the input to the hash function.

5.2.1.3 Video Frame Hashing Cost Estimation

For each video frame added to a bucket, a new node is added to the bucket's linked list to store the video sequence identifier, the frame occurrence position of the video frame, the numbers of video frames in the video sequence, and the date the video sequence was captured.

The total cost to hash one video sequence can be estimated as follow:

$$CH(m) = \sum [H(n) + L(n)] \quad \text{for } n = 1 \ldots m \qquad \text{(Equation 5.13)}$$

where

m = size (total video frame count) of the video sequence

H(n) = a fixed cost of one basic hashing pass

L(n) = linked list traversal cost

Referring to equation 4, H(n) is a fixed cost for hashing key generation and table lookup, while L(n) depends on the size of the linked list, or the total number of video

sequences containing a same similar video frame. With a video window size of 1.296 million frames (i.e., 15 frames per second for 24 hours), the worst case for CH(m) is when all the 1.296 million frames are similar. In this case, the video hash table will contain only one bucket with a linked list of 1.296 million elements. On the other hand, the best case for CH(m) will occur when the 1.296 million frames are unique. In other words, the more unique video frames, the less total video hashing cost is affected by L(n). The collision rate of 2 to 1 predicted by our simulation experiment predicts that the linked lists will, on average, remain short. As we will see later in the result discussion section, this is validated in practice and the linked list traversed does not affect the overall hashing performance. We also verify that the total hashing cost is independent of the total video window size, and hence is a fixed cost linearly proportional to the size of the input video sequence (m).

5.2.2 Video Sequence Filtering Process

If, for all frames in the sequence, there are no matching video frames in the hash table, we can stop the video identification process and conclude that the new input video sequence is a new video sequence. This is based on the assumption that if both the input video sequence and video archive have no similar frames in common, then we can say that there should be no repetitive connection between them and hence they must be different. If there are some video frame matches between the input video and

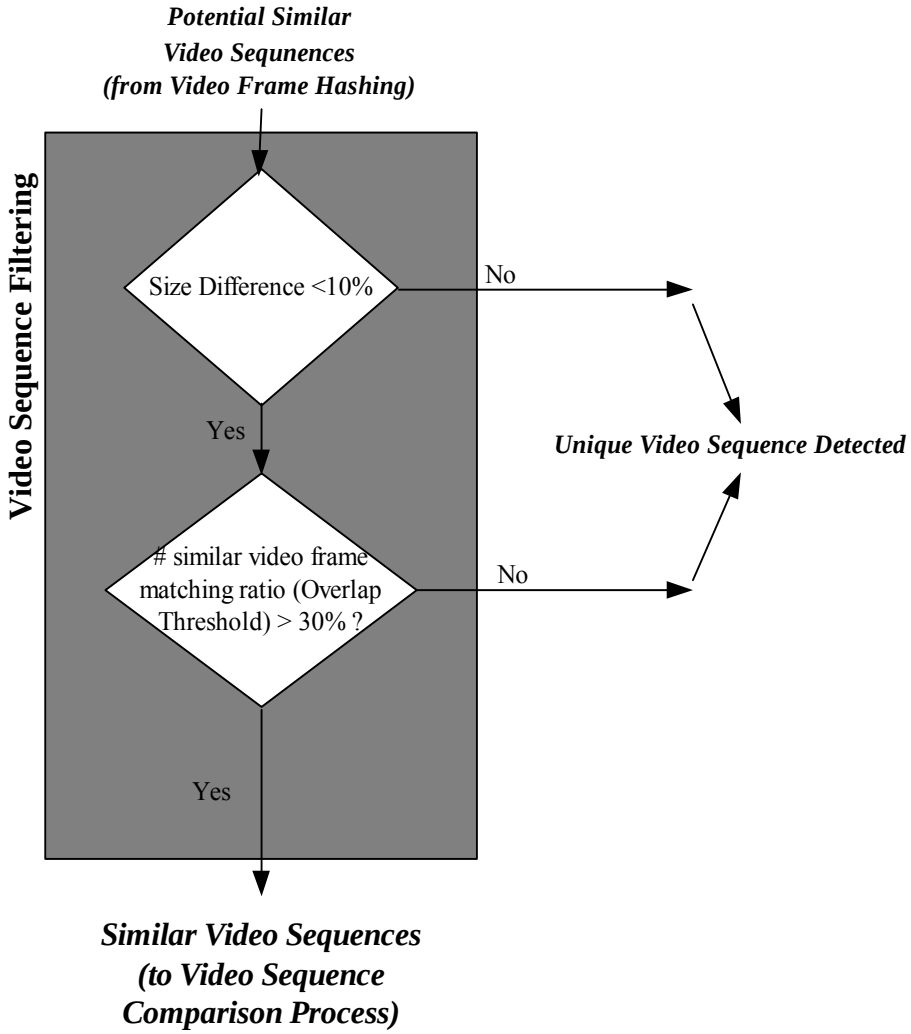


Figure 5.9 Flow diagram of Video Sequence Filtering

the archive, we must develop a heuristic to determine how much overlap is required to identify a similar sequence. We consider two video sequences to be candidates for similarity (or similar video sequences) if the following two conditions are met:

1) the size difference of the matching video sequences is less than 10%, and

2) the total number of matching video frames is at least 30% of the total number frames of smaller video sequence.

If no video sequences survive this filtering process, then the video sequence input is identified as a new video sequence (see Figure 5.9). Video sequences pass this filtering process are considered "similar", but they are not necessarily identical to the matching input video sequence. The video sequence comparison process then will be used to determine if these qualified similar video sequences are indeed repeats of the matching input video sequence.

In summary, the main purpose of the overall video hashing process is to decrease the number of video sequence comparisons required by efficiently identifying video sequences in the video archive that one similar to the input sequences. Using this video hashing technique, we were able to reduce the video identification computation time required during normalized video sequence comparison tremendously (see Section 5.3.3). An example of video hashing process is illustrated in Appendix D. The pseudocode for the final overall video hashing process, including the video sequence filtering, is shown below:

/*** ***Final Video Hashing Process Pseudocode*** ***/

Process Input *: A linked list containing color moment string for each video frame of the input video sequence*

/ **Component 1** - Similar Video Frame Hashing */*

for (each color moment string of the input video sequence)

{

Do hash bucket index number generation;

Do hash table hashing;

If (Hits)

Collect meta-information of video sequences & accumulate occurrences of color moment string matching;

Next color moment string – video frame;

}

/ **Component 2** - Video Sequence Filtering */*

/ potential similar sequence : video sequence having at least one similar video frame to the input video sequence */*

for (each potential similar video sequences captured)

{

Remove the video sequence having size difference > 10% when compared to the input video sequence;

Apply overlap threshold to screen out "dissimilar" video sequences;

}

***Process Final Output**: A linked list containing similar video sequences passing both the size difference and hashing threshold screening checks*

5.3 Video Sequence Comparison

The final step of our video sequence identification algorithm performs a more accurate repeated video sequence identification using a frame-by-frame comparison technique illustrated in Figure 5.10. This is to eliminate false positives created during the video hashing process caused by color moment approximation and ignorance of temporal ordering of the matching video sequences. The absolute moment differences between video frames from the input video sequence and the similar video sequences are calculated. The result is compared with a moment difference threshold to determine whether or not the video sequences are similar enough for the input video sequence to be considered a repeat sequence. Appendix D contains a detailed example of the video sequence comparison process.

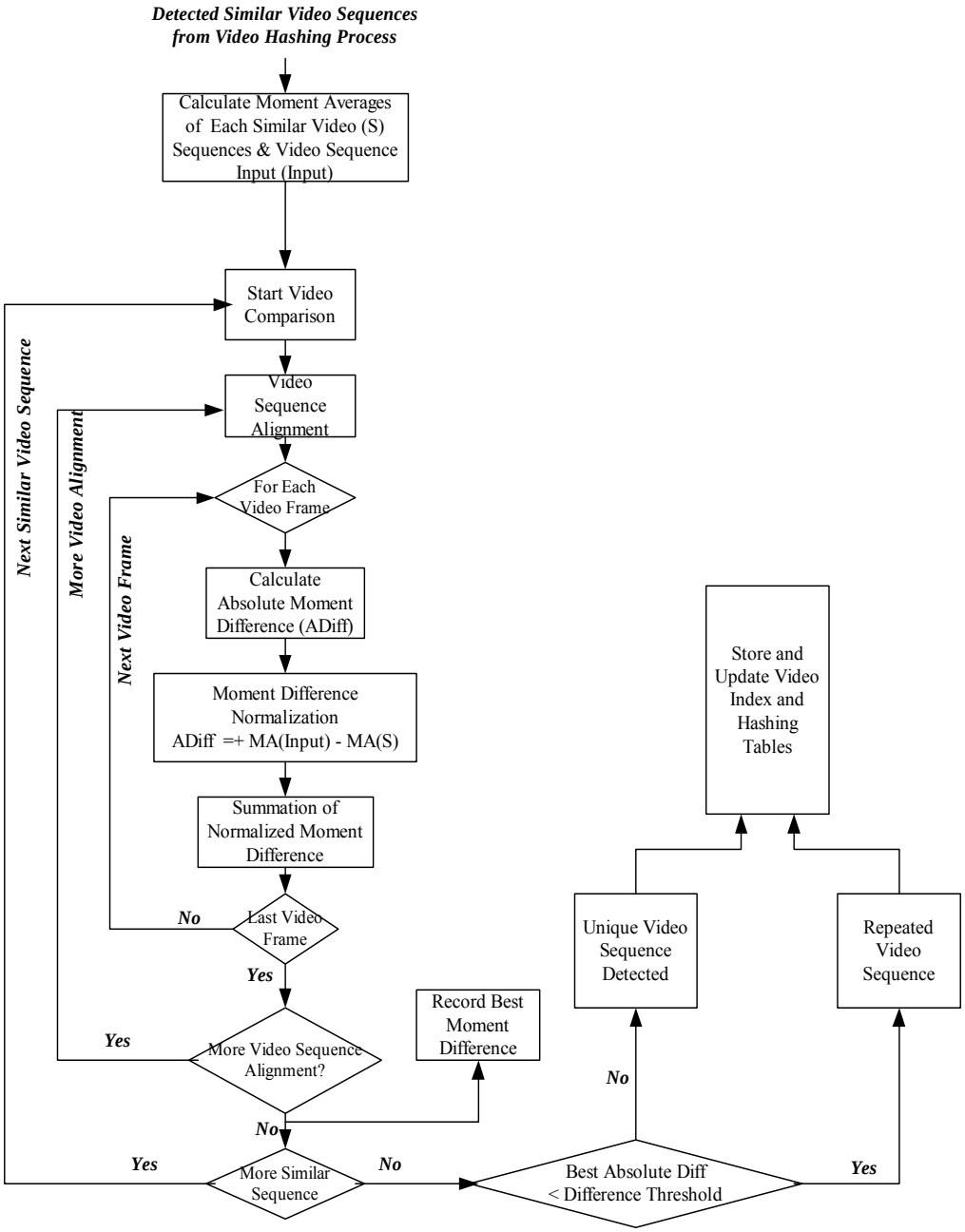


Figure 5. The Flow Diagram of Video Sequence Comparison Process

Figure 5.10 The flow diagram of the Video Sequence Comparison Process

The following pseudocode describes steps performed during the video sequence comparison process:

*/*** **Complete Video Sequence Comparison Process** ***/*

Process Input :

1) *The Input Video Sequence*
2) *A list containing similar video sequences, Similar[M]*

```
Output[];
Best Detected Repeated Sequence=NULL;
Best Moment Difference =1000;  // captures the best moment difference
M = Total number of captured similar video sequences;

/* compare each similar sequence to the input video sequence*/
for ( x=0;x< M, x++))
 {
        Video Sequence Comparison Function(Input, Similar[x], Difference, Best
        Frame Pos);
        If (Difference < Best Moment Difference)
         {
         Best Moment Difference = Difference;
        Best Sequence = Similar[x];
        }
        /* Insert Similar Sequence into Output[], sorted by Difference */
        Insert(*Output, Similar[x], Difference, Best Frame Pos);
}
/* Moment Difference Threshold screening */
if (Best Moment Difference < Moment Difference Threshold)
        Detected Repeated = Best Sequence;
```

Process Output:

1) Detected Repeated Sequence;

2) List of similar sequences with respective moment differences

5.3.1 The Absolute Moment Difference Calculation

During video sequence comparison, the absolute moment difference between video frames from both the input video sequence and similar video sequences are calculated. The absolute moment difference of two video sequences is calculated as follows:

Moment Average of a video sequence for each moment number (*c= 1…9)*:

$$MomA(c) = \frac{1}{N} \sum_{n=1}^{N} m(c,n)$$

where

m(c,n) = cth color moment in the nth frame,

N=size of video sequence

c = 1….9 (color moment) (Equation 5.14)

Normalized Absolute Moment Difference for each moment number:

$$Diff_c(v1,v2) = \frac{1}{N} \sum_{n=1}^{N} (| m_{v1}(c,n) - m_{v2}(c,n) - [MomA_{v1}(c) - MomA_{v2}(c)] |)$$

where

v1 & v2 are video sequences being compared

$m_{v1}(c,n)$ = c moment value of the nth frame n in video sequence v1

N = sequence size of the smaller video sequence of v1 & v2

n = video frame temporal order

c = color moment index (1..9)

(Equation 5.15)

Sum Absolute Moment Difference:

$$MomDiff(v1, v2) = \frac{1}{M} \sum_{c=1}^{M} Dif_c(v1, v2)$$

where

M = total moment numbers (9)

v1 & v2 are the two video sequences being compared

c = color moment index (1…9)

$Dif_c(v1,v2)$ = normalized absolute moment difference of sequence v1 & v2 for color moment c

(Equation 5.16)

The absolute moment difference value calculated between similar video sequences is checked against a moment difference threshold (see Chapter 6 for experimental results) to determine if it is close enough to the input to be considered repeated. The following pseudocode describes the flow of the absolute moment difference calculation function designed and implemented as part of the overall video comparison process:

/** Video Sequence Comparison Function - Part of VSI Module****/***

Function Input*:*

1) Original color moments of the input video sequence

2) Original color moments of a similar video sequence –output of the video hashing process

/ calculate moment average for each sequence – See Equation 5.13 */*

Calculate average value of each 9 moments in the input video sequence ;

Calculate average value of each 9 moments in the similar video sequence;

/ calculate sum absolute moment difference for each possible sequence alignment */*

For (each sequence alignment shifting)

{

For (each comparing frame)

{

Calculate normalized absolute moment difference

- See Equation 5.14

}

Accumulate Sum Absolute Moment Difference - See Equation 5.15

If (calculated new sum absolute difference < previous recorded sum absolute difference)

{

Best Difference = new sum absolute difference;

Record the new sequence alignment position;

}

}

Function Output:

1) The best difference value

2) The sequence alignment position creating the best difference value

5.3.2 Aligning Video Sequence for Best Comparison Result

Video Segmentation error due to noise might result in two repeated video sequences having different video sequence size. So, we use a simple video sequence alignment method (Figure 5.11) during the video sequence comparison to align the two video sequences being compared into the proper position to achieve the best sum absolute difference result. Sum absolute moment different is calculated for each of these alignments, and the best value is recorded. We allow a total shifting of up to 10% of video sequence size. Referring to Figure 5.11, the alignment process begins by aligning the start of the larger sequence to the *n*th frame of the smaller sequence, in which *n* frame is one tenth of the size of the smaller sequence. We slide the smaller sequence, one frame at a time, to the right until it reaches a point where the last frame of the larger sequence is aligned to the mth frame of the smaller sequence, in which case the *m*th frame is 90% into the size of the smaller sequence (see Figure 5.11). The sum of absolute moment difference for each alignment is calculated and the best value is recorded.

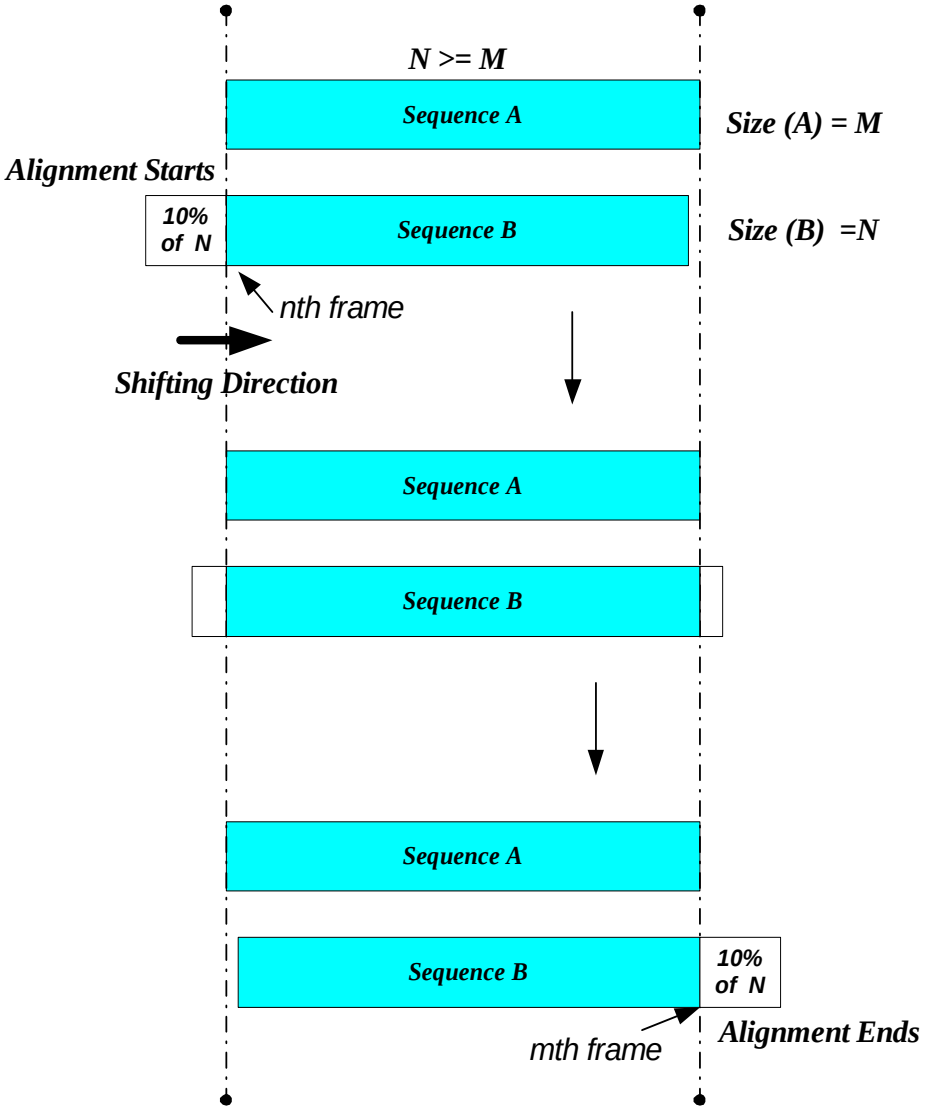


Figure 5.11 Video sequence alignment flow chart

The following pseudocode describes the alignment process implemented as art of the video comparison function:

```
/*** Video Sequence Alignment Process ***/
Process Input:  Comparing Sequence A[M] and comparing sequence B[N]  & M<N

Size Dif = (Size A) – (Size B);

/* Calculate the 10% size of the smaller sequence */
AlignShift = (Size A)/ 10;

/* find the proper frame position for both A & B for sequence comparison
  shift one frame at a time after each video comparison */
For (shiftframe= -AlignShift; shiftframe < (Size Dif +AlignShift); shiftframe++)
{
        /* calculating correct start and end frame position of each sequence
        alignment */
        If (shiftframe >0 && shiftframe< Size Dif)
        {
                ShiftframeA =0;
                shiftsize =0;
                shiftframeB=0;
        }
        if (shiftframe <0)
        {
                shiftframeB =0;
                shiftsize = shiftframe;
                shiftframeA = -shiftframe;
```

```
        }
        if (shiftframe > Size Dif )
        {
                shiftsize = Size Dif – shiftframe;
                shiftframeA = 0;
                shiftframeB = shiftframe;
        }

        /* now have the proper frame position for both sequences A & B,
          do sequence comparison */
        for (frame=0; frame < Size(A) + shiftsize; frame++)
       {
                /* calculate Absolute Moment Difference  */
                Temp =  A[frame+shiftframeA] – B[shiftframeB+frame];
                Do Moment Difference Normalization;
                Accumalte Moment Difference;
        }
        /* Record Sum Absolute Difference for this alignment */
        Sum Difference;
        Record best frame position;
        Go to next alignment;
}
```

***Process Output**: Best Sum Absolute Moment Difference of this two sequence comparison*

5.3.3 Video Sequence Comparison Cost Estimation

The total video comparison cost for one video sequence input, *x*, can be estimated as follow:

$$TotalCost(x) = \sum_{n=1}^{M} AllCompare(n, S_n, x)$$

where

M = Total number of similar video sequences detected

S_n = size of n^{th} video sequence (Equation 5.16)

$$AllCompare(n, S_n, x) = 0.1 * S_n * O[OneCompare(n, x)]$$

(Equation 5.17)

O[OneCompare(n,x)] represents the total one pass cost of summation of the absolute moment difference between two video sequences and hence is linearly proportional to the size of both video sequences. Equation 5.17 estimates the total cost of the video comparison by a allowing video sequence alignment adjustment of 10% of the size of the smaller video sequence. Both equations show that the comparison process is heavily dependent on the number of similar video sequences and sequence size. Therefore, it is our goal to reduce this total video comparison cost by reducing the total numbers of similar video sequences (*M*) detected in the video hashing process.

5.4 Video Sequence Archiving and Tracking

The process of video sequence archiving and tracking works hand-in-hand with the video sequence identification process. While the video sequence identification process is the core function for the VSI module, video sequence archiving and tracking is the responsibility of the VSA module. For each input video sequence, the sequence identification or matching is performed in the VSI module. Results from the video identification are used by the VSA module to record the sequence's meta-information, extracted by the VPS module, and its video identification outcome. The VSA module will also decide whether or not to save the encoded video clip into the VSS module, based on whether or not it is identified as a new sequence.

5.4.1 Video Sequence Index Table

Figure 5.12 illustrates the video index table data structure used while tracking and indexing captured video sequences. The Video Sequence Index Table consists of a video index header and a video index array. Each array element contains a data structure called *Input Sequence Information* that is used to store the meta-information of an input video sequence and two pointers pointing to two data elements (*Repeated Sequence Reference* and *Repeat Video Linked List*).

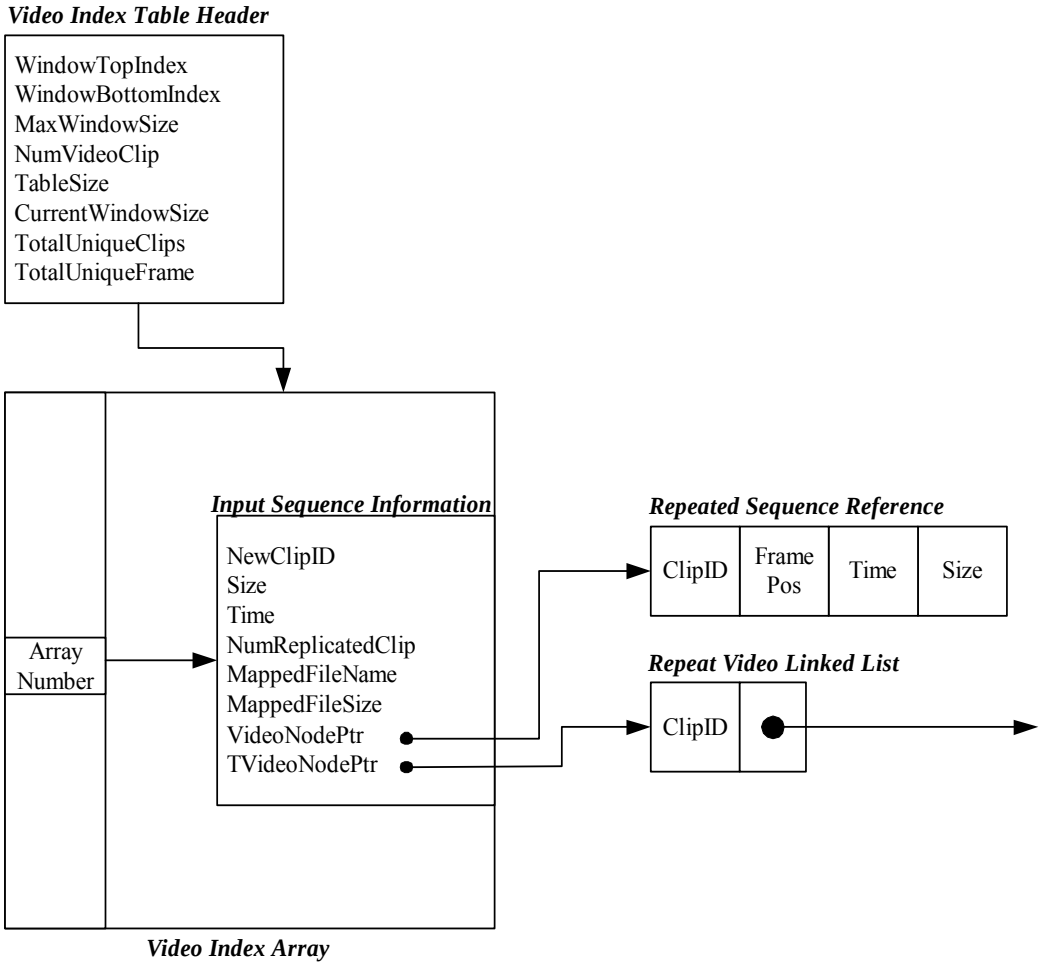


Figure 5.12 Video sequence index table

1)Video Index Header

Table 5.2 contains the definition of each parameter found in the video index header. The video index header keeps track of the overall process information for the video index table such as numbers of sequences captured and total unique sequences detected.

Variables	**Definition**
WindowTopIndex	Video Array index number of the 'oldest' video sequence captured
WindowBottomIndex	Table index number of the next available video index array element
MaxWindowSize	The allowable maximum size of the video archive (Sliding Video Window Size)
NumVideoClip	Total number of video sequences stored
TableSize	Size of the video index array
CurrentWindowSize	Total size of the video archive
TotalUniqueClips	Total number of unique sequence captured
TotalUniqueFrames	Total number frames of the unique sequences captured

Table 5.2 Video index header parameter definition

2) Input Sequence Information Data Structure

The Input Sequence Information data structure found in each array element is used to store the meta-information of each input video sequence. In the other words, we will need to have one entry of the Input Sequence Information data structure for every

input video sequence. Table 5.3 contains the definition of each parameter found in the Input Sequence Information data structure. For each Input Sequence Information data structure of an input video sequence, the Repeated Sequence Reference data element (see Figure 5.12) is used to record information of an "old" sequence if the input video sequence stored in this Input Sequence Information data structure is detected as a repeat of the "old" sequence. The Repeat Video Linked List is a sorted linked list that is used to record information for repeat occurrences for the input video sequence

Variable	**Definition**
Array Number	The video array index number
NewClipID	The video sequence identifier
Size	The size of video sequence
Time	The year video sequence is captured
NumReplicatedClip	Repeated sequence indicator. 1 means repeated sequence, 0 otherwise
MappedfileName	Filename of Real video sequence file
MappedFileSize	Size of Real video sequence file
VideoNodePtr	Pointer to Repeated Sequence Reference data element
TvideoNodePtr	Pointer to a linked list containing information of related repeated video sequences

Table 5.3 Video index array parameter definition

For every pass of input video sequence identification, the index information of the input video sequence is stored and indexed based on whether it is identified as a repeat video sequence or as the first occurrence of a new video sequence. If the input video sequence is considered a new sequence, then the Repeated Sequence Reference data element will be set to NULL. If the input video sequence is detected as a repeat, then the detected 'older' matching video sequence will be identified as the input sequence's Repeated Sequence Reference and its information will be stored in the Repeated Video Sequence Reference data element. The information kept in the Repeated Video Sequence Linked List is needed in order to select a new Repeated Sequence Reference whenever an expired video sequence is removed during the sliding window process. The parameters in the video index header are updated based on the outcome of the video identification process.

Figure 5.13 illustrates a flow diagram of the video indexing process. Referring to the figure, the main function of the indexing process is to store and keep track of meta-information and video sequence identification results of input video sequences. Another main function of the indexing process is to control and enable the video sliding window mechanism by keeping track of the total video sequences processed and their captured time stamps. Since the video index array was implemented as a fixed-size array structure, we use an array wrap-around indexing mechanism.

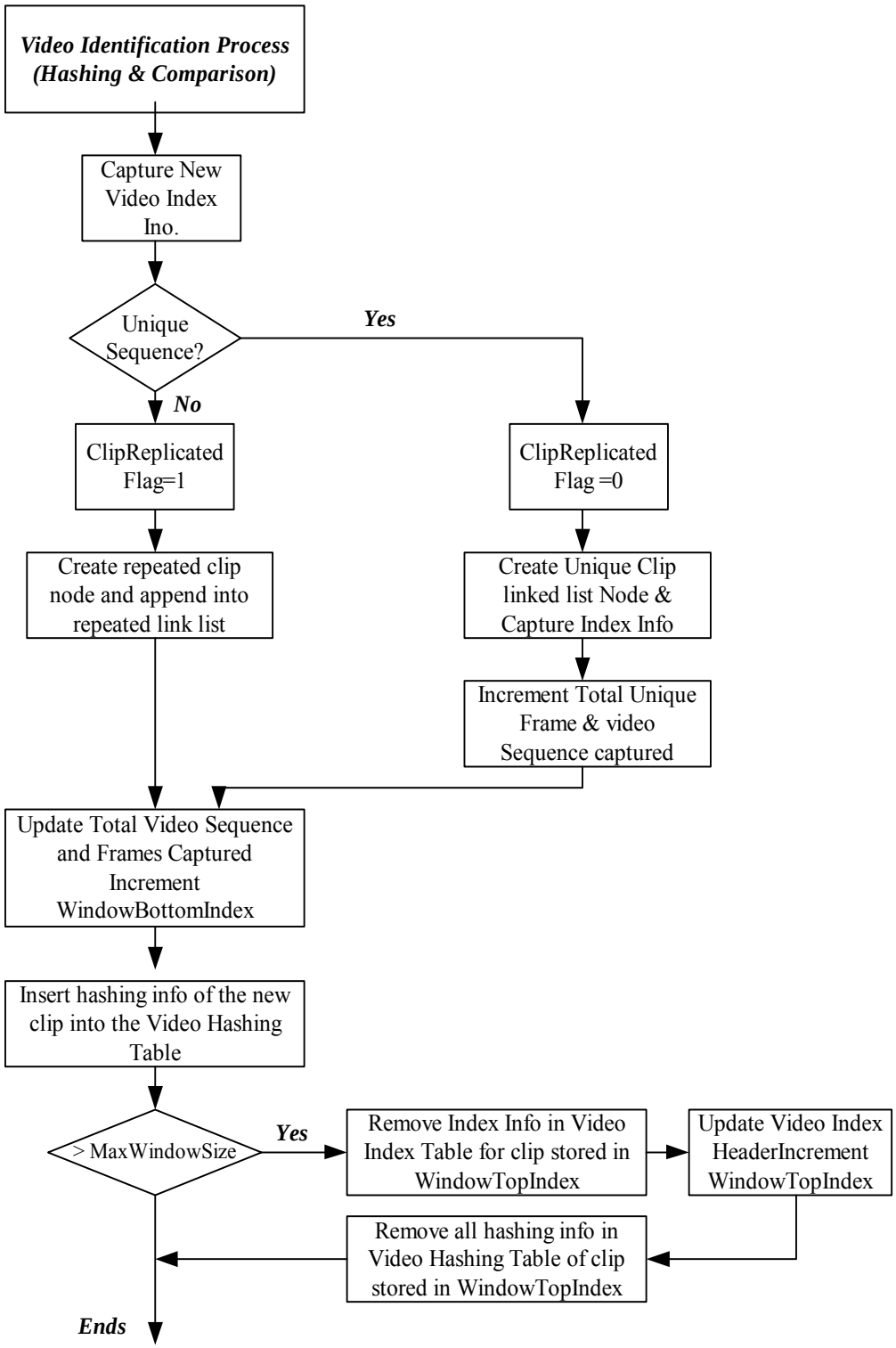


Figure 5.13 The flow diagram of the video indexing process

5.4.2 Inserting Index Information of a Input Video Sequence

Whenever the Video Sequence Identification Module (VSI) completes the processing for an input video sequence, the results and the meta-information for that video sequence is saved in the video index table. Color moment strings of the input video sequence are also inserted into the video hash table.

5.4.2.1 Video Index Table Insertion

The index information for the input video sequence is stored in the *WindowBottomIndex* (next available array element) (see Table 5.2 for definition). We store the meta-data in the Input Sequence information data element of the assigned array element. If the input video sequence is considered to be new, the Repeated Sequence Reference data element and the Repeat Video Linked List are set to NULL. If the input video sequence is considered to be a repeat, we insert its index information into its respective Repeated Sequence Reference data element. The Repeat Linked List of the selected Repeated Sequence Reference sequence is also updated. The Repeat Linked List of a video sequence in an array element keeps track of indices for the 'newer' video sequences that are considered to be repeats of the sequence recorded in this array element. These newer video sequences consider the video sequence recorded in this array element as their Repeat Sequence Reference. Updates of related parameters in the video index table complete the input video sequence insertion. An example of video index table insertion is illustrated in Appendix D.

The following pseudo code describes the overall video sequence insertion process:

*/*** Video Index Table Insertion Process***/*

***Process Input**:*

1) *Input Video Sequence meta-information*
2) *Input Video Sequence Identification Results*

ArrayIndex = WindowBottomIndex;
Copy meta-information and matching results into array elements;
/ **Unique Sequence Detected** */*
If (Input Video Sequence is New – Unique)
{
    Set NumReplicatedClip =0;
    Set Repeated Sequence Reference element to Null; // a new unique sequence
    Set Repeat Video Linked List to Null; // no related sequences exists yet
    Move sequence encoded clip into the Video Sequence Storage (VSS);
    TotalUniqueClip++;
    TotalUniqueFrames += Size of the Input Video Sequence;
}
/ **Repeated Sequence Detected** */*
If (Input Video Sequence is Repeated)
{
    Set NumReplicatedClip =1;
    Insert the detected matching video sequence into the Repeated Sequence Reference data element; // matching is found
    Search array index of the detected matching sequence;

Insert the input video sequence into its Repeat Video Linked List (sorted by sequence identifiers)

Remove encoded sequence file from the input common folder

}

NumVideoClip++;

CurentWindowSize += Size of the Input Video Sequence;

WindowBottomIndex++;

Record results into respective log files;

***Process Output**:*

1) *Video Index Table Update*
3) *Log files recorded*

5.4.2.2 Video Hash Table Insertion

Video hash table insertion involves the insertion of color moment strings of the input video sequence into the video hash table. For each color moment string of the input video sequence, calculate the hash bucket value containing the matching hashing key (the color moment string) and insert the sequence index information into the linked list pointed by the hash bucket. If there is no matching color moment string, find the next available hash bucket and insert the color moment string into the hash table and its sequence index information into its respective linked list. Appendix D illustrates an example of the hash table insertion process.

5.4.3 Deleting Index Information of An Expired Video Sequence

We implemented a sliding window mechanism (Figure 5.14) to allow the total video archive to grow until it contains 24 hours of sequences from video stream, after which the oldest sequences are dropped as the newer sequences are added. The sliding window mechanism was implemented on a wrap-around fixed-size array. Whenever an expired video sequence is removed from the video archive, two things must be done: 1) delete array element of the expired video sequence and update the sequence index information of the repeated sequences related to the expired video sequence; and 2) delete color moment strings of the expired video sequence from the video hash table.

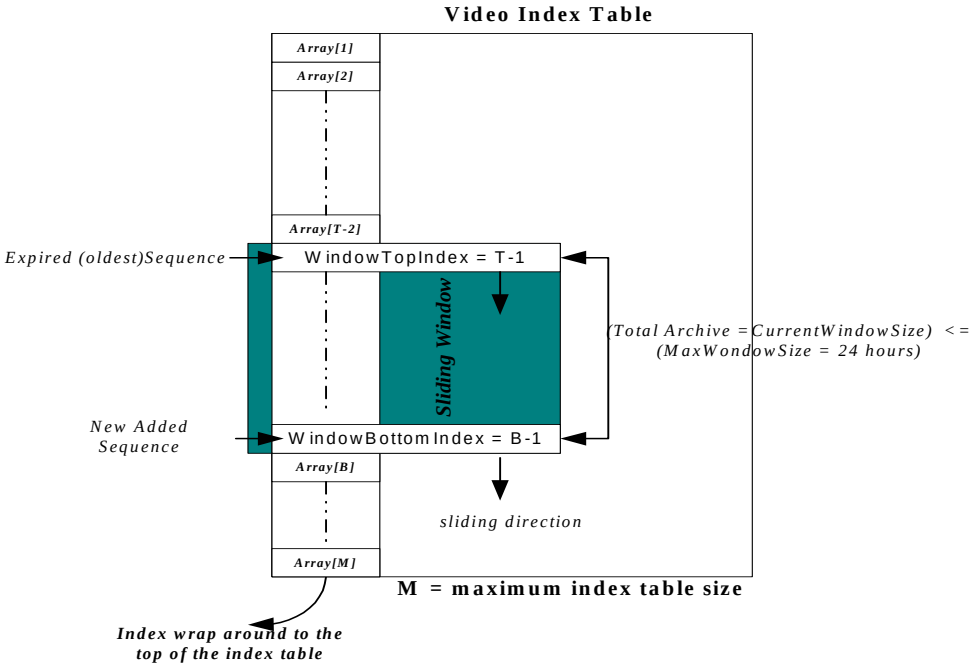


Figure 5.14 Video hash table sliding window mechanism

5.4.3.1 Deleting An Expired Video Sequence

Whenever an expired video sequence is removed from the video index table, the index information of other video sequences in the video archive relating this expired video sequence should be updated. There are two types of video sequences update scenarios:

a) The expired sequence was a unique sequence with no repeated video sequences

Since the expired video sequence is the only instance of this sequence in the video archive and there are no 'newer' video sequences that are repeats related of it, the system merely needs to erase the data stored in the array element indexed by WindowTopIndex (see Table 5.2 for definition) and increment WindowTopIndex to complete the sliding window process. The total number of unique sequences is decremented by 1.

b) The expired sequence was new and has at least one related repeat video sequence

If the removed expired video sequence was the first occurrence of video sequence that has at least one related repeat video sequence, (Repeat Video Linked List is *not NULL*), then the system needs to appoint a new Repeated Sequence Reference sequence for all the repeat sequences found in the Repeat Video Linked List. We use the first sequence from the time ordered Repeat Video Linked List (the oldest sequence among the sequences recorded in the Repeat Video Linked List) as the new Repeated Sequence Reference sequence for all the others. Hence, we reset both the

Repeated Sequence Reference data element and the Repeat Video Linked List of the newly selected Repeated Sequence Reference to NULL. For each remaining sequence in the Repeat Video Linked List of the expired sequence, we change their Repeated Sequence Reference to the newly assigned reference sequence. At the same time, we append the information of these remaining sequences into the Repeat Video Linked List of the new Repeated Sequence Reference sequence. The total number of unique sequences stored in the final video archive does not change. Finally, we remove the expired sequence from the array element and increment WindowTopIndex to complete the sliding window process. Please refer to Appendix D for an illustration of the expired sequence removal process.

The following pseudo code describe the deletion process of a video sequence in the video index table:

```
/*** Video Sequence Deletion from the Video Index Table ***/
Process Input:  1) The Expired Video Sequence

Expired Sequence Array Element = WindowTopIndex;

/* Video Archive Overflow Checking */
If (CurrentWindowSize > WindowSize )  // < 24 hours
{
 /* The expired sequence is new and has related repeat sequences*/
   if  (Expired ->Repeated Sequence Reference == NULL ) //Expired Sequence is new
      if (Expired->Repeat Video Linked List != NULL) // related repeat sequences
     {
```

```
                /*Select first element of Repeated Linked List as the
                 new Repeated Sequence Reference */
                New Reference  = Expired->Repeat Video Linked List[0];
                New Reference->NumReplicatedClip =0;

                // redirect encoded file index informaton
                New Referecne->MappedFileName= Expired->MappedFileName;
                New Reference->MappedFileSize = Expired->MappedFileSize;

                // redirect matching results
                New Reference->Repeated Sequence Reference = NULL;
                New Reference->Repeat Video Linked List = NULL;

                /*reset related sequences' repeat(matching) reference*/
                For (each remaining sequences in Expired Sequence->Repeat Video
                Linked List) {
                        Remaining->Repeated Sequence Reference = New Reference;
                        Append(New Reference->Repeat Video Linked List,
                        Remaining);
                }
        }


        TotalUniqueClips--;
        TotalUniqueFrames -= Size of the Expired Sequence;
  }

NumVideoClip--;
WindowTopIndex++;
```

CurrentWindowSize -= Size of the Expired Sequence;

Remove Expired sequence from the table;

}

***Process Output**: Remove Expired Video Sequence from Video Archive (VSA)*

5.4.4 Deleting Color Moment Strings of An Expired Video Sequence

Whenever an expired video sequence is removed from the video archive, the previously inserted color moment strings of the sequence should also be removed from the video hash table. This process involves hashing into the video hash table to locate the correct bucket and then traversing the linked list to remove the element in which the expired sequence is stored. See Appendix D for an illustration of expired color moment string removal.

Chapter 6

Experimental Results and Discussion

We evaluated our video sequence identification technique by measuring the efficiency and accuracy of the technique and the storage compression achievable for the video archive using a test collection of 32 hours of continuous video stream captured from a television documentary channel. Segmentation of this video stream created 2,831 video sequences. To evaluate the system, we submitted each sequence to the identification software. Each sequence is treated as a query against the video abstraction archive to see if any matching sequences can be found. The archive is allowed to grow until it contains 24 hours of sequences from video stream, after which the oldest sequences are dropped as the newer sequences are added. Viewing and tracking the video sequences manually produced a "ground truth" of 1,228 new video sequences and 1,603 repeats against which our algorithm is compared.

6.1 Measuring the Video Sequence Identification Accuracy and Efficiency

6.1.1 Video Hashing Time

We first studied the relationship between the size of the Video Sequence Abstraction's hash table and the lookup time needed to find a similar sequence in the hash table. The video sequence hashing time is calculated as the sum of the hash table lookup time during video frame hashing processing and the time measured during video sequence filtering process. The video sequence hashing time for two different repeated sequences (one 895 frames long, the other 377 frames long) are shown in Figure 6.1. Each data point represents the hash table lookup and video sequence filtering time for one occurrence of the repeated video sequences for a given video window size (or hash table size). The shorter sequence occurred eight times. Its lookup and filtering took from 25ms (with a video archive size of 10,000 frames) to 80ms (with a video archive size of 1,100,000 frames), with an average of 40ms. The longer sequence occurred seven times. Its lookup and filtering took from 55ms (with a video archive size of 900,000 frames) to 130ms (with a video archive size of 1,000,000 frames), with an average of 85ms. The results show that, as expected, the time taken is independent of the size of the hash table. This experimental result is consistent with the mathematically derived total video hashing cost (see section 5.2.1.3), in which the total video sequence hashing cost can be estimated as the sum of a fixed hash table lookup and a fixed linked list traversed cost that is linearly

proportional to the size of the input video sequence (40ms for shorter sequence versus 85 ms for longer sequence).

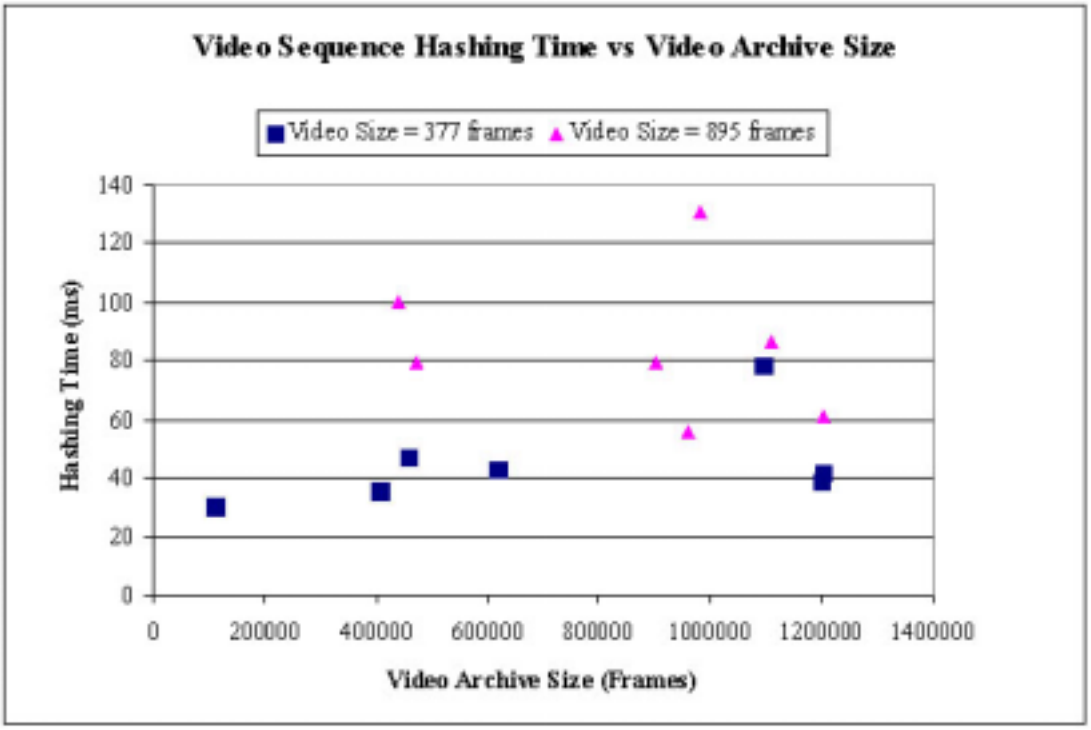


Figure 6.1 Graph of video sequence hashing time versus video archive size

In order to better understand the effect of video sequence size on video sequence hashing time, we ran an experiment to measure the video sequence hashing time for nine video sequences of various sizes using the same video archive size. Figure 6.2 shows the video hashing time for these nine video sequences. Each data point represents the video sequence hashing time for a video sequence. Referring to the Figure, a video sequence with a size of 1,000 frames has a video hashing time of close to 100ms, followed by a video hashing time of 500ms for a sequence with a size of 2,800 frames. The results in this figure show that the total video hashing time

increases gradually with the video sequence size. In the other words, the experimental results shown in this figure suggest that the video hashing time for repeated video sequences is linearly proportional to the length of the video sequence. Although we capped the video archive size at 24 hours (1.296 million video frames) for our later experiments, these results imply that we could increase the video window size to support longer time periods without affecting the speed of the video hashing process.

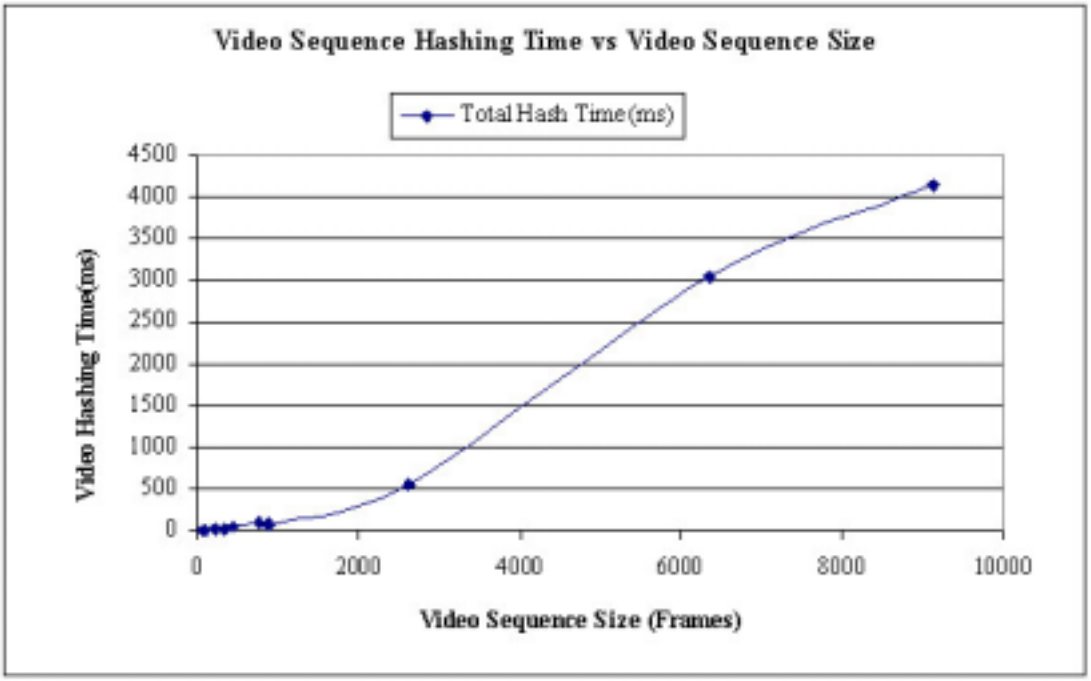


Figure 6.2 Graph of video sequence hashing time versus video sequence size

6.1.2 Video Sequence Comparison Cost

The final step in our comparison process compares the input video sequence to the similar video sequence abstraction(s) frame by frame to calculate sum of the absolute moment different of the two sequences. Figure 6.3 shows the comparison times for 10 sequences where the sequence length varies from 1,000 frames to 10,000 frames. Referring to the figure, a sequence of 1,000 frames requires a comparison time of 400ms and a sequence size of 2,000 framers requires a comparison time of 550ms. In other words, there is an increase of about 150ms in comparison time. On the other hand, a sequence size of 9,000 frames requires a comparison time of 20,500ms and a sequence size of 10,000 frames requires a comparison time of 25,200ms. So, the increase of comparison time between sequence with 9,000 frames and sequence with 10,000 frames is 4,700ms. As expected, these results indicate that the video comparison cost increases drastically as the video sequence size increases, growing faster than linear as sequence size gets larger. Also, the cost for this process is greater than the video sequence hashing process. This indicates that, to keep the entire repeated sequence identification cost low, we should use meta-data (e.g., sequence length) and a overlap threshold during the video sequence filtering process to keep the number of similar video sequences low by filtering out obviously dissimilar video sequences. We should also keep the video sequences short, preferably segmenting at least every 200 – 400 seconds (3,000 – 6,000 frames).

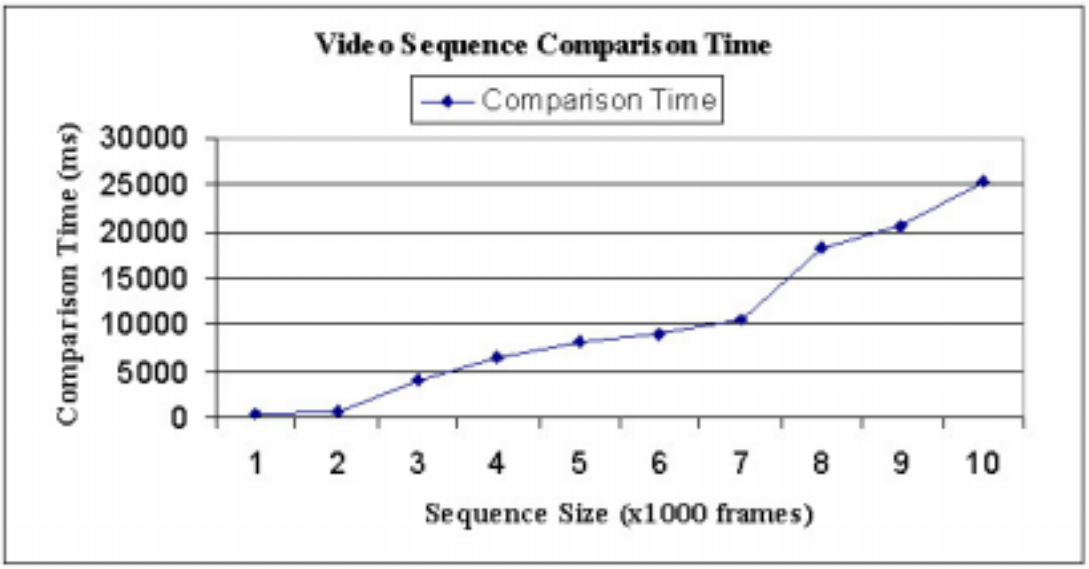


Figure 6.3 Graph of video comparison time versus video Sequence size

6.1.3 Measuring Recall and Precision

We measure the accuracy of our results using true positives (correctly identifying the input video sequence as a repeat occurrence of an already stored video), false positives (incorrectly identifying a stored video sequence as a match for the input video sequence), true negatives (correctly identifying the input video sequence as the first occurrence of a new video sequence) and false negatives (incorrectly identifying a input video sequence as a new video sequence when it is actually a repeat of a video sequence stored in the archive). These numbers are combined to calculate the two traditional information retrieval metrics: 1) *Recall*, the ratio of the number of correctly detected repeated sequences to the total number of repeated sequences in

test; and 2) *Precision*, the ratio of the number of correctly detected repeated sequences to the number of detected repeated sequences in the video archive.

$$Recall = true\ positives / (true\ positives + false\ negatives)$$

$$Precision = true\ positives / (true\ positives + false\ positives)$$

Appendix E demonstrates an example of how true positive, true negative, false positive, false negative are collected from the outputs of the video sequence identification process to calculate recall and precision.

6.1.4 Choosing An Optimum Overlap Threshold Value

A stored video sequence is considered similar to the input video sequence if the percentage of the frames in the two sequences with the same color moments exceeds the overlap threshold. As the value of overlap threshold is raised, fewer video sequences are considered similar, and therefore the total video comparison cost is decreased. However, higher overlap thresholds may result in more misses (i.e., false negatives) of repeated video sequences.

We ran an experiment to evaluate the effect of varying the overlap threshold value to examine the tradeoff between speed and accuracy. Each video sequence is submitted to the video hashing process and all similar sequence matches are reported. If no similar sequence matches are found, then the input video sequence is treated as the first occurrence of a new video sequence. If there are similar sequence matches, the input video sequence is considered a repeat sequence of these detected similar

sequences. This experiment measured how well the video hashing process performs under various overlap threshold values.

Overlap Threshold (%)	0	10	20	30	40	50	60	70	80
True Positive	6108	6082	6044	5953	5779	5439	4781	3600	1886
False Positive	49734	4788	2264	1648	1154	833	505	263	126
True Negative	202	872	1067	1148	1194	1245	1346	1552	1963
False Negative	222	248	286	377	551	891	1549	2730	4444
Recall	0.96	0.96	0.95	0.94	0.91	0.86	0.76	0.57	0.30
Precision	0.11	0.56	0.73	0.78	0.83	0.87	0.90	0.93	0.94
Average Hashing Time Per Sequence(ms)	541	540	544	536	549	522	521	494	518
Average Sequence Compare Time Per Sequence(ms)	8575	986	597	547	533	530	486	482	446

Table 6.1 Recall and precision versus a set of different overlap threshold values

Table 6.1 shows the recall and precision measured for video hashing process for various values of the overlap threshold. The table captures the four numbers (true positive, true negative, false positive, false negative) used to measure the recall and precision. As shown in this table, both true positives and false positives decrease with the increase of the overlap threshold value. This result is expected since the tighter the similar sequence check is (higher hashing threshold value), the more true repeated video sequences are mistakenly removed during the similar video sequence filtering process, and hence fewer true positives are found. Since fewer similar video sequences exceed the overlap threshold, the number of false positives (mistakenly detecting a first occurrence sequence as repeated sequence) is reduced. On the other hand, true negatives and false negatives increase with the increase of the overlap threshold value. This is also expected since fewer similar video sequences exceed the

higher overlap threshold, resulting in more true negatives (identifying the input video sequence as the first occurrence of a new video sequence) and also higher numbers of false negatives (identify the input video sequence as the first occurrence of a new video sequence when it is actually a repeat).

The results shown in Table 6.1 indicate that an overlap threshold value of 30% yields a good tradeoff between accuracy and efficiency and it is used in our later experiments. Compared to a overlap threshold of 0%, recall drops from 96% to 94% yet the total comparison time is cut tremendously (8,575 ms versus 547 ms). The data indicates that the average video hashing time is independent of the value of overlap threshold used in selecting similar sequences. The data also indicates a minor decrease in the average sequence comparison time is possible with a higher overlap threshold value, but recall is adversely affected. For example, a threshold of 60% will decrease comparison time 12% (to 486ms from 536ms) but recall declines 20% (to 0.76 from 0.94).

6.1.5 Video Sequence Comparison Accuracy

The previous experiments show that while an overlap threshold can be used to improve speed and precision, precision remains somewhat low (78%). To remove false positives and thereby increase precision, we compare the surviving similar video sequence abstractions to the input video sequence abstraction frame by frame. Only if this comparison exceeds a moment difference threshold is the input video sequence considered a repeat occurrence of a sequence in the archive.

To evaluate the effect of the moment difference threshold on repeat sequence detection accuracy, we measured the recall and precision for a variety of moment difference thresholds and the results are shown in Table 6.2. The recall and precision measured in video sequence hashing process (indicated as Initial) is included in this table for comparison purposes. As expected, higher moment difference thresholds result in higher recall (fewer false negatives) but lower precision (more false positives). The results suggest that a moment difference threshold of 10.0 represents a reasonable balance between recall and precision. Comparing to recall and precision recorded for video sequence hashing process without the video sequence comparison, precision improves from 78% to 91% due to the removal of false positives. Also, there is a slight drop in recall from 94% to 91% due to incorrect removal of a few true positives. The results show that precision can be improved by performing the video sequence comparison to remove false positives at the expense of increasing false negatives and slightly decreasing recall.

Moment Difference Threshold	Initial	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	15.0	20.0
Tpositive	5953	4090	4756	5134	5456	5598	5743	5809	5831	5889	5917
Fpositive	1648	314	385	438	532	580	595	627	656	741	1039
Tnegative	1148	1498	1406	1358	1326	1299	1273	1259	1247	1222	1179
Fnegative	377	2240	1574	1196	874	732	587	521	499	441	413
Recall	0.94	0.65	0.75	0.81	0.86	0.88	0.91	0.92	0.92	0.93	0.93
Precision	0.78	0.93	0.93	0.92	0.91	0.91	0.91	0.90	0.90	0.89	0.85

Table 6.2 Recall and precision measurement (Overlap Threshold=30)

There were a total of 2,831 video sequences created from the test collection of 32 hours continuous video stream. Using at a 24 hour sliding window of stored video abstractions, a perfectly operating system would identify 1,228 new sequences and 1,603 repeat sequences. Our system, with an overlap threshold of 30 and a moment difference threshold of 10, was able to identified 1,225 new sequences and 1,553 repeat sequences.

6.2 Measuring the Achievable Storage Compression Ratio

Since repeated video sequences need only be stored once, our video sequence repetition identification can be used to achieve lossless storage compression. Since this compression does not manipulate the video signal in any way, it can be used in addition to other compression techniques. Figure 6.4 shows the number of sequences in the video archive versus number of detected new sequences for the entire identification process. The compression gain can be calculated as the ratio of total size for repeated video sequences to the total size of video sequences stored in the video archive. Clearly, for a video source that contains repeated sequences, the larger the video archive, the more repeats are likely to be found. Figure 6.5 shows that compression increases as the size of the video archive is increased, up until the 1,261st sequence (approximately 13 hours), at which time it levels off at approximately 30% compression. With a video window size of 24 hours, the video source used in this experiment contains an approximation of 8 hours of repeat programs during the 32 hours span. This data suggests that if our video identification technique was able to

correctly detect all repeated video sequences, we should be able to get a compression gain of 33%.

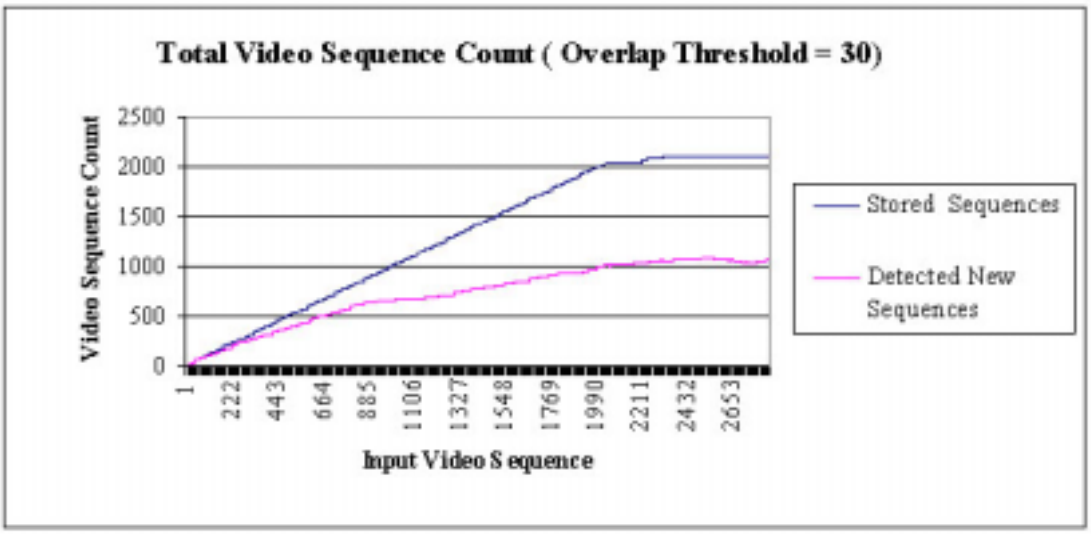


Figure 6.4 Total video sequence in the video archive vs. total detected repeated sequences

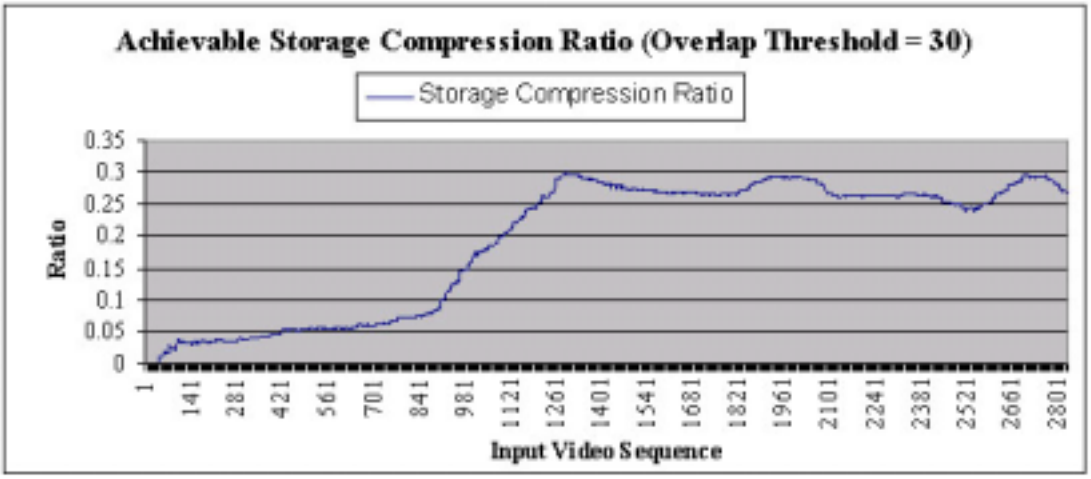


Figure 6.5 Ratio of achievable lossless storage Compression

6.3 Validating Video Identification Technique with Different Video Source

Our previous experiments allowed us to identify an overlap threshold (30%) and a moment difference threshold (10.0) that provides accurate repeated video sequence identification. To validate these results, we re-ran our algorithm using 24 hours of video (3,394 video sequences) collected from a different broadcast television channel. Using a sliding window of 24 hours, if the system worked perfectly, it would identify 1,938 unique video sequences and 1,456 repeats. Table 6.3 reports the recall and precision measurements for different values of a moment difference threshold using an overlap threshold of 30%. With an overlap threshold of 30 and a moment difference threshold of 10, our system was able to identify 1,903 out of 1,903 true new sequences and 1,385 out of 1,456 true repeated sequences.

Moment Difference Threshold	Initial	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	15.0	20.0
Tpositive	5341	4148	4666	4953	5119	5198	5219	5236	5245	5325	5341
Fpositive	1324	188	246	308	331	358	371	401	440	613	865
Tnegative	1833	2141	2068	2042	2011	1978	1973	1967	1961	1913	1878
Fnegative	554	1747	1229	942	776	697	676	659	650	570	554
Recall	0.91	0.70	0.79	0.84	0.87	0.88	0.89	0.89	0.89	0.90	0.91
Precision	0.80	0.96	0.95	0.94	0.94	0.94	0.93	0.93	0.92	0.90	0.86

Table 6.3 Recall and precision measurements with overlap threshold of 30

The moment difference threshold of 10.0 provides comparable performance on this second video source. Recall with the second source is slightly lower (89% versus

91%), however precision is slightly higher (93% versus 91%). These consistent results validate that our technique and threshold settings generalize to other video sources.

Figure 6.6 captures the total number of sequences in the video archive versus the total cumulative detected new sequences for the entire identification process. Figure 6.7 shows that we were also able to achieve a compression ratio of 30%, comparable to the compression ratio on the first video source. Similar to the first video source, the second video source used in this experiment has approximately of 8 hours of repeat programs over 24 hours span. With a perfect video identification technique, we would get a compression gain of approximately 33%.

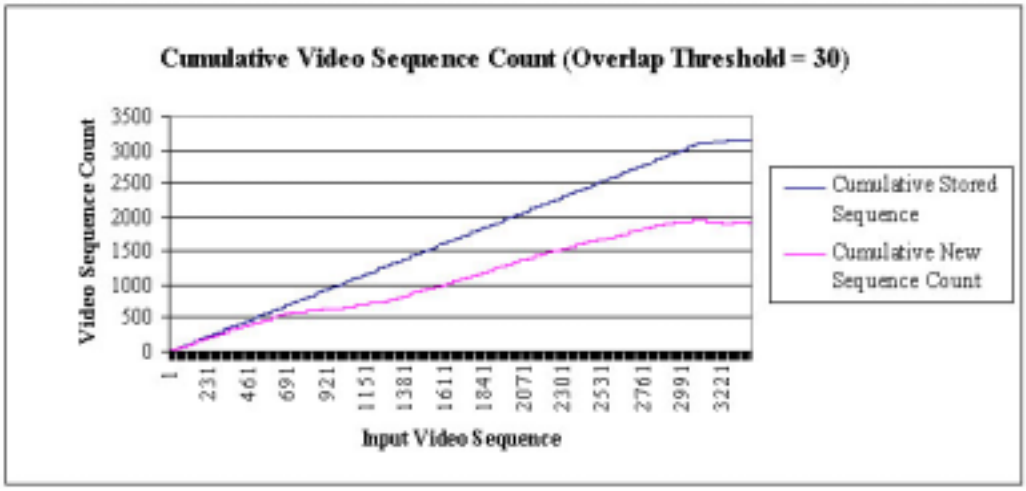


Figure 6.6 Total sequences in video archive vs. total detected new sequences (Overlap Threshold=30 & Moment Difference Threshold =10.0)

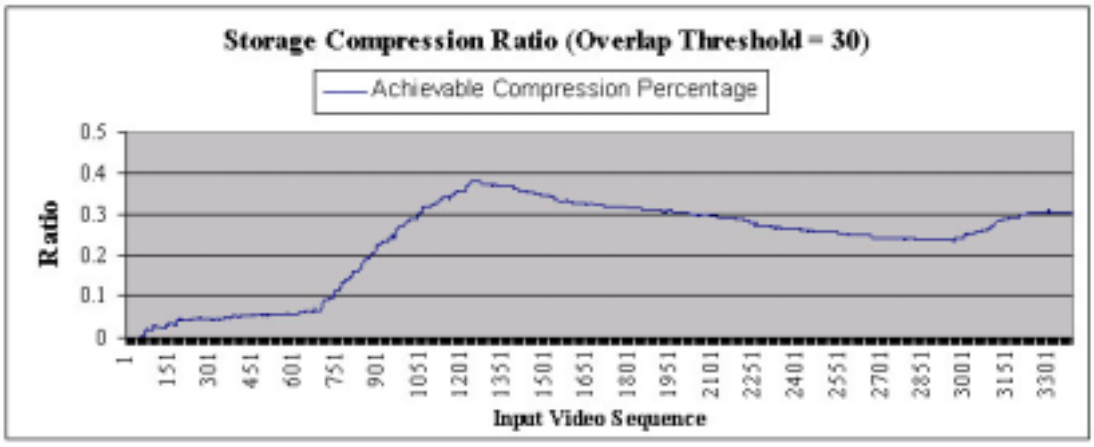


Figure 6.7 Total achievable storage compression ratio

Chapter 7

Conclusion

7.1 Summary

This dissertation reports on an automatic video sequence identification and tracking technique that can be used to process continuous video streams, identify unique video sequences and remove repeated video sequences. The technique described here is efficient (it runs in real time) and effective. Our technique is domain and video source independent so that they could be used on any video streams that are repeated and change slowly over time.

We have developed a system to digitize and segment video streams into video sequences using histogram-based techniques. Each video frame of video sequences is represented by values of the nine color moments, namely the mean, standard deviation, and the skew of the three main color components. The Color Moment String, created by mapping the nine color moment values into nine integers and

concatenating them into a string, is used as the basis of our video sequence identification technique.

Our video sequence identification solution employs the combination of similar video sequence hashing and frame by frame video sequence comparison to detect and identify repeated video sequences. In the similar video sequence hashing process, each video frame is represented by a color moment string. The similarity of two video sequences is measured by considering the percent of frames in the two sequences that have similar video frames ignoring the temporal order of the similar frames. The purpose of the similar video sequence hashing process is to reduce the size of the number of sequences requiring frame by frame matches by selecting only video sequences which have many similar frames.

During frame-by-frame video sequence comparison, each video frame is represented by the full values of the nine color moments extracted during the video processing and stream segmentation process. The sum absolute moment differences between video frames from the input video sequence and the similar sequences identified by the video hashing process are calculated. The absolute moment difference value calculated for each similar video sequence is compared with a moment difference threshold to determine whether or not the video abstractions are similar enough for the input video sequence to be considered a repeated sequence. In this video sequence comparison process, each pair of video frames is compared in their temporal order.

The total cost of performing a video sequence comparison using our video sequence identification is the sum of the hashing process and the absolute color moment difference calculation during video sequence comparison process. The experimental results suggested that the cost of our video sequence identification technique increases logarithmically with video archive size and hence is able to handle a large video archive size.

Using a maximum video abstraction archive size of 24 hours, we evaluated our approach on two different video continuous streams, (one 32 hours long and the other 24 hours long). We were able to achieve good recall and precision (over 90%) on both inputs and hence validated the accuracy of our technique. The experimental results measured on both inputs also justified our claims that the technique is domain and video source independent.

We also evaluated the achievable video archive storage compression by measuring the total amount of video data consisting of repeated video sequences. By not storing the repeated video sequences, we achieved a lossless compression gain factor of approximately 30% for both video streams.

Finally, this system can be used as the first step of a topic tracking system for video streams and/or to compress the viewing time needed for end users, allowing them to quickly find out “what’s new”.

7.2 Future Work

Future work falls in two broad categories: 1) technique improvement, and 2) user application development.

7.2.1 Technique Improvement

a) Partial Mapping We can improve the quality of the tracking by incorporating partial mapping. We get a partial match whenever one video sequence partially overlaps another. The processed video sequence could be a subset of a known video sequence or it could be a superset of few video sequences overlapping one another. Our current video identification algorithm was designed in such a way that it can be easily extended to support the partial matching of video sequences. One of the main algorithm enhancements will be to include adjustable video sequence size difference screening during the similar video sequence filtering process, allowing similar video sequence matching for difference sizes. The final video sequence comparison process will then be divided into three steps: 1) compare video sequences with the same size (size difference is less than 10%) for exact match; 2) compare video sequences that are longer than the input video (i.e., consider the input as subset of a known video sequence); and 3) compare video sequences that are smaller than input video sequence (i.e., the input as superset of known video sequences).

b) Disk based hashing In order to handle a very large video window size, a disk based hashing technique should be used. A disk based hashing technique will allow the video identification algorithm to grow to larger video archive, for example seven days, to detect and track repeated video sequences that occur farther apart in time.

7.2.2 User Application

a) Web-enabled Video Stream Browsing System A web-enabled client-server-based graphical user interface could be built on top of our system to enable users to search the video archive and view selected video sequences. The system can be used to compress the viewing time needed for end users, allowing them to quickly find out "what's new". The system will support dynamic reproduction of video stream for select broadcast using the video sequence tracking information captured in the video identification process.

b) Story based Video Sequence Identification This research work could form one component of a video archive system for content-based topic tracking. Figure 9.1 illustrates a function block diagram for an ideal model of fully automated video content-based story tracking system for television news programs. In order to detect and aggregate unique news stories from a video source, we have to have a fully automated video signal processing system that can first apply content-based video processing technique to track and extract all the unique video sequences from the input source. Then video sequences must be grouped into different stories using

video abstractions such as closed caption, audio and video content. Hence, the next possible step of our research could be designing story-based video sequences identification technique by combining semantic information extracted from both video content and a possible text input. We could be concentrating on the knowledge intensive activity of content-based aggregation into stories.

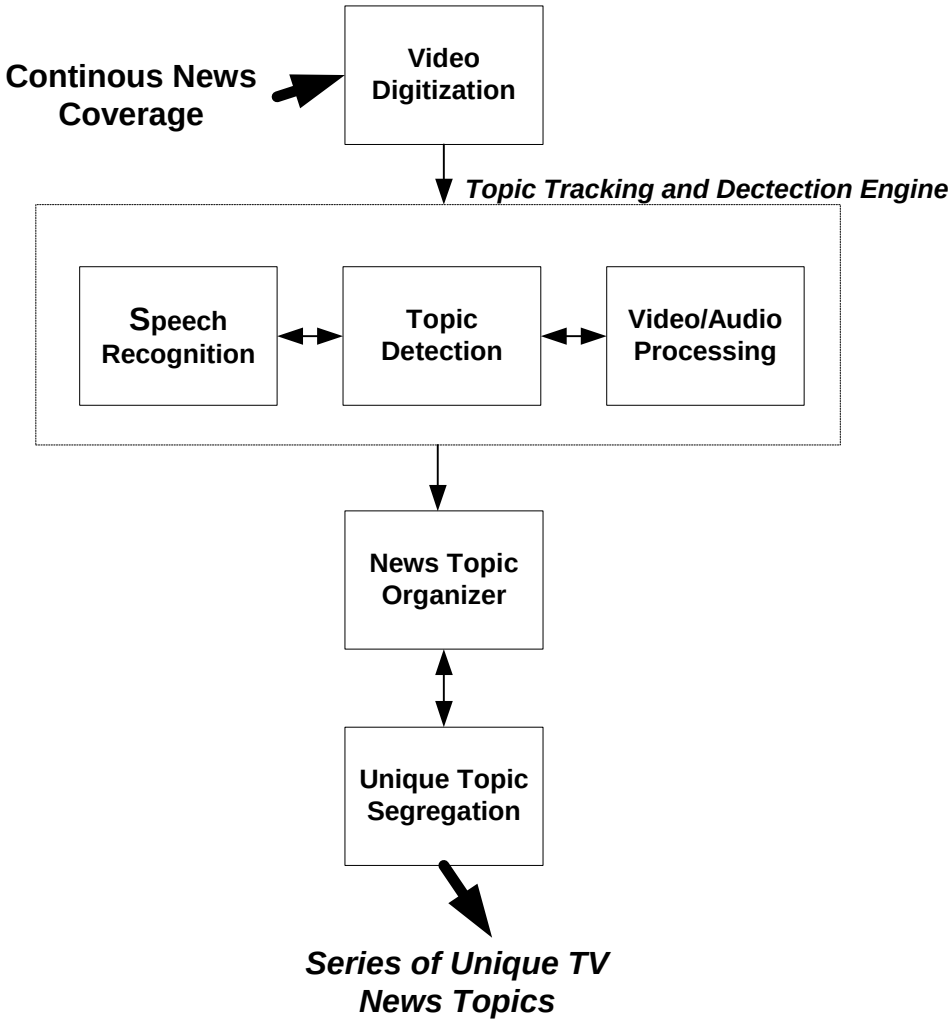


Figure 7.1. Functional block diagram of a ideal television news topic tracking system

Bibilography

[1] Allan, J., Carbonell, J., Doddington, G., Yamron, J., and Yang, Y., (1998). Topic Detection and Tracking Pilot Study Final Report. In Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop, February, 1998.

[2] Allan, J., Lavrenko, V., and Jun, H., (2000). *First Story Detection In TDT Is Hard*. In ACM CIKM 2000, pages 374-381.

[3] Arman, F., Depommier, R., Hsu, A., and Chiu, M.Y., (1004). Content-based Browsing of Video Sequences. In ACM Multimedia 1994, San Francisco.

[4] Bouix, S. (1998). VISION: Segmentation, Indexing and Retrieval of Digital Videos. Master Thesis of EECS, The University of Kansas.

[5] Chueng, S.C., and Zakhor, A., (2000). Efficient Video Similarity Measurement and Search. In International Conference on Image Processing Vol 1, pp. 85-89, British Columbia.

[6] Chueng, S.C., and Zakhor, A., (2000). Estimation of Web Video Multiplicity. In Proceeding of The SPIE on Internet Imaging Vol 3964, pp.34-46, San Jose.

[7] Dagtas, S., McGee, T., and Mohamed, A.M. (2000). SmartWatch: An Automated Video Event Finder. In ACM Multimedia 2000, Los Angeles.

[8] Dimitrova, N., (1997). Content-based Video Retrieval by Example Video Clip. In the Proceeding of SPIE on Storage and Retrieval for Image and Video Database V, 1997, Vol. 3022. San Jose.

[9] Dimitrova, N., (1995). The Myth of Semantic Video Retrieval. In ACM Computing Surveys, Vol 27, No. 4, December 1995

[10] Fiscus, J., Doddington, G., Garofolo, J., and Martin, A., (1998). NIST's 1998 Topic Detection and Tracking Evaluation (TDT2). National Institute of Standards and Technology, Gaithersbury, MD, 1998.

[11] Flickner, M., Sawhney, J.A., Huang, Q., Dom, B., Gorkani, M., Lee, D., Petkovic, D., Steele, D., and Yanker, P., (1995). Query by image and video content: The QBIC System. In IEEE Computer, 28(9):23-32, Spetember 1995.

[12] Gauch, J., (2000). Video Authentication: Overview. http://www.ittc.ukans.edu/~jgauch/research/video/vidwatch_overview.html

[13] Gauch, J., Gauch, S., Bouix, S., Zhu, X., (1999). Real Time Video Scence Detection and Classification, Information Processing and Management 33 '99.

[14] Gauch, S., Gauch, J., Pua, K.M., (1996). VISION : A digital Video Library, ACM Digital Libraries '96, Bethesda, MD, 19-27

[15] Gauch, S., Gauch, J., Pua, K.M., (1998). The VISION Digital Video Library Project, Encyclopedia of Library and Information Science, '98.

[16] Gauch, S., Li, W., Gauch, J., (1997). The VISION Digital Video Library System, Information Processing & Management, 33(4), 413-426.

[17] Geusebroek, J.M., Koelma, D., and Smeulders, A.W.M., (2000). Image Retrieval and Segmentation Based On Color Invariants. In Computer Vision and Pattern Recognition 2000, Hilton Head, South Carolina.

[18] Huang, J., Kumar, S.R., Mitra, M., (1997). Combining Supervised Learning With Color Correlograms For Content-Based Image Retrieval. In ACM Multimedia 1997, Seatle.

[19] Huang, J., Kumar, S.R., Mitra, M. et al., (1997). Image Indexing Using Color Correlograms. In Proceedings of Computer Vision and Pattern Recognition, page 762-768, 1997.

[20] Jain, A.K., (1989). Fundamentals of Digital Image Processing. Prentice Hall, USA 1989.

[21] Kobla, V., Doermann, D., (1997). VideoTrails: Representing and Visualizing Structure In Video Sequences. In ACM Multimedia 1997, Seattle.

[22] Kato, T., et al (1992). A Sketch Retrieval Method for Full Color Image Database: Query by Visual Example. In Proc. 11th International Conference on Pattern Recognition, Amsterdam, Holland, 1992.

[23] Lienhart, R., Effelsberg, W., and Jain, Ramesh. (1998). VisualGREP: A Systematic Method to Compare and Retrieve Video Sequences. In the Proceedings of SPIE on Storage and Retrieval for Image and Video Database VI, Vol. 3312. Jan. 1998.

[24] Ogle, V., and Stonebraker, M., (1995). Chabot: Retrieval from a relational database of images. In IEEE Computer, 28(9):40-48, September 1995.

[25] Pan, J.Y., and Faloutsos, C., (1999). VideoGraph: A New Tool for Video Mining and Classification.

[26] Pass,G., and Zabih, R., (1998). Comparing Images Using Joint Histograms. ACM Journal of Multimedia System, 1998.

[27] Pass, G., and Zabih, R., (196). Histogram Refinement for Content-Based Image Retrieval. In Proceedings of the 3rd IEEE Workshop on Applications of Computer Vision,1996.

[28] Pentland, A., Picard, R., and Sclaroff, S., (1996). Photobook: Content-based manipulation of image databases. In International Journal of Computer Vision, 18(3):233-254, 1996

[29] Pua, K.M., Gauch, S., Gauch, J., (1999). VIDSEEK: Dynamic Multi-dimensional Browsing of Video Archives, Multimedia Indexing and Retrieval Workshop, ACM SIGIR '99, Berkeley, CA, USA.

[30] Qian, R., Haering, H., and Sezan, I., (1999). A Computational Approach to Semantic Event Detection. In IEEE Computer Vision and Pattern Recognition, June 1999, Fort Collins, Col.

[31] Rao., A., Srihari, R.K., Zhang, A., (1999). Spatial Color Histogram For Content-Based Image Retrieval. In 11th IEEE International Conference on Tools with Artificial Intelligence,1999, Chicago.

[32] Ray, B., (1998). Analysis of Closed Caption Based Segmentation of Digital Video, Master Thesis, The University of Kansas, KS.

[33] Rickman, R., and Stonham, J., (1996). Content-based Image Retrieval Using Color Tuple Histograms. In SPIE Proceedings, 2670:2-7, 1996

[34] Rowe, L.A., Boreczky, J.S., and Eads, C.A., (1994). Indexes for User Access to Large Video Databases. In Proc. IS&T/SPIE Conf. On Storage and Retrieval for Image and Video Database II, pp. 150-161, San Jose, CA, 1994.

[35] Rui, Y., Huang, T.S., and Mehotra, S., (1998). Exploring Video Structure Beyond The Shots. In Proceedings of The IEEE International Conference on Multimedia Computing and Systems, 1998.

[36] Smith, J., and Chang, S.F., (1996). Tools and techniques for Color Image Retrieval. In SPIE Proceedings, 2670:1630-1639, 1996

[37] Stricker, M., and Dimai, A., (1996). Color Indexing with Weak Spatial Constraints. In SIPE Proceedings , 2670:29-40, 1996.

[38] Swain, M.J., and Ballard, D.H., (1991). Color Indexing. In International Journal of Computer Vision Vol 7, pp. 11-32, 1991.

[39] Tanveer, S.M., and Srinivasan, S. (2000). Detecting Topical Events in Digital Video. In ACM Multimedia 2000, Los Angeles.

[40] Wu, Y., Zhuang, Y., and Pan, Y., (2000). Content-based Video Similarity Model. In ACM Multimedia 2000, Los Angelas, CA

[41] Yoon, K., DeMenthon, D., and Doermann, D. (2000). Event Detection from MPEG Video in the Compressed Domain. In Proceedings of the International Conference on Pattern Recognition 2000.

[42] Zhang, H.J., Low, C.Y., Smoliar, S.W., and Wu, J.H., (1995). Video Parsing, Retrieval and Browsing: An Integrated and Content-Based Solution. In ACM Multimedia 1995, San Francisco, CA.

[43] Zhang, H.J., Tan, S.Y., Smoliar, S.W., and Gong, Y.H., (1995). Automatic Parsing and Indexing of News Video. In Multimedia System 1995, Vol 2: 256-266.

Appendix A

Statistical Model of Image Texture Representation

I. The Autocorrelation Function (ACF)

The width of the spatial *ACF* $r(k,l) = m_2(k,l) / m_2(0,0)$ represents the spatial size of texels in the texture. $m_2(k,l)$ is the second moment or the mean square value or average energy and is defined as follow:

$$m_2(k,l) = \frac{1}{N_w} \sum_{(m,n) \in w} [u(m-k,n-l)]^2$$

The coarseness of texture is expected to be proportional to the width of the ACF which can be represented by distances x_0, y_0, such that $r(x_0, 0) = r(0,y_0) = ½$. The calibration of the ACF spread on a fine-coarse texture scale depends on the resolution of the image. This is because a seemingly flat region (no texture) at a given resolution could appear as fine texture at a higher resolution and coarse texture at lower resolution. Therefore, the ACF by itself is not sufficient to distinguish among

several texture fields because many different image ensembles can have the same ACF.

II. Image Transforms

Texture features such as coarseness, fineness, and orientation can be estimated by generalized linear filtering techniques utilizing image transforms (Figure A.1). Image transforms provide the frequency domain information in the data. Transform features are extracted by zonal-filtering or feature masking the image in the selected transform space. Referring to Figure A.1, a two-dimensional transform *V(k,l)* of the input image is passed through several band-pass filter *g(k,l).* The energy in this *V^(k,l)* represents a transform feature.

The feature mask is simply a slit or an aperture shown in Figure A.2. Generally, the high-frequency features can be used for edge and boundary detection, and angular slits can be used for detection of orientation. Fore example, an image containing several parallel lines with orientation θ will exhibit strong energy along a line at angle $\pi/2 + \theta$ passing through the origin of its two-dimensional Fourier transform. The combination of these two types of masks is useful for periodic or quasi-periodic textures. Image transforms have been applied for discrimination of terrain types such as deserts, farms, mountains, and riverbeds to name a few.

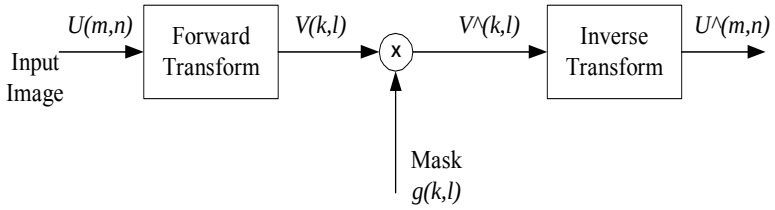
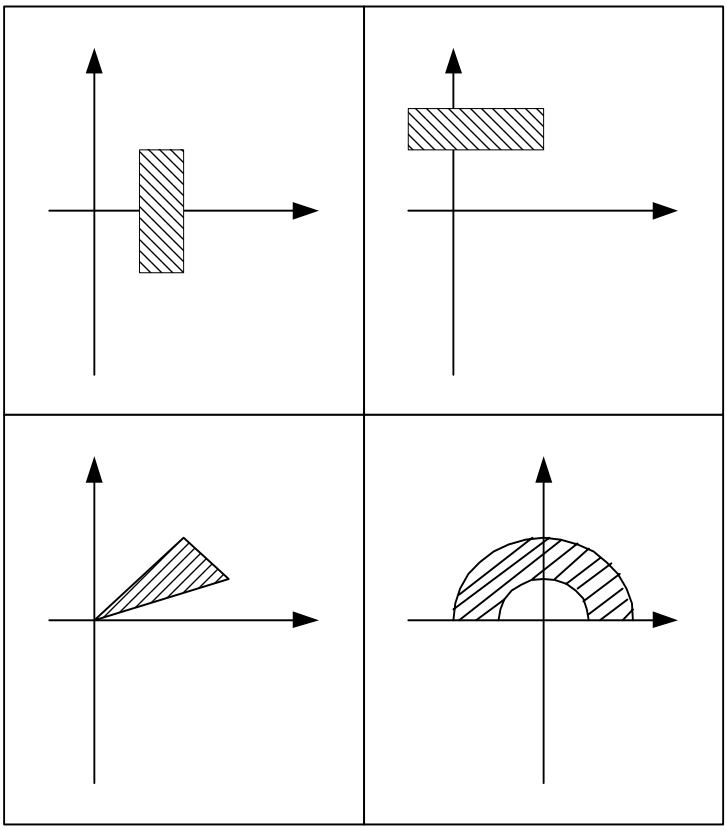


Figure A.1 Transform Feature Extraction



Slits and Apertures

Figure A.2 Slits and Apertures

III. Histogram Features.

Some useful texture features based on the histogram measures are:

$$Inertia: I(r,\theta) \cong \sum_{x_1}\sum_{x_2} \left||x_1 - x_2|\right|^2 f(r,\theta; x_1, x_2)$$

$$Mean: \mu(r; x_1, x_2) = \frac{1}{N_0} \sum_{\theta} f(r,\theta; x_1, x_2)$$

$$Variance: \overset{2}{\sigma}(r; x_1, x_2) = \frac{1}{N_0} \sum_{\theta} [f(r,\theta; x_1, x_2) - \mu(r; x_1, x_2)]^2$$

$$Spread: \eta(r; x_1, x_2) = \max_{\theta}\{f(r,\theta; x_1, x_2)\} - \min_{\theta}\{f(r,\theta; x_1, x_2)\}$$

where $f(r,\theta;x_1,x_2)$ is the distribution function of two pixels x_1 and x_2 at relative distance r and orientation θ. The inertia is used to represent the spread of the function $f(r,\theta;x_1,x_2)$ for a given set of (r,θ) values. $I(r,\theta)$ becomes proportional to the coarseness of the texture at different distances and orientations. The mean distribution is useful when angular variations in textural properties are unimportant. The variance indicates the angular fluctuations of textural properties while spread distribution is used to measure the orientation-independent spread.

Appendix B

Moment Invariant Measurements

The moment of a gray-level image *f(x,y)* is defined as:

$$m_{pq} = \quad x^p y^q f(x,y)$$

If we considered the gray-level as the weight of the pixel, m_{00} can be viewed as the total mass of the image, m_{10} and m_{01} the centroids, and m_{20} and m_{02} would represent the moments of inertia around the *x* and *y* axes. Moment invariants are calculated with the following steps:

1. Computer central moments:

$$\eta_{pq} = \quad (x-\bar{x})^p (y-\bar{y})^{(q} f(x,y)$$

where $\bar{x} = m10 / m_{00}, \bar{y} = m_{01} / m_{00}$

2. Normalize them:

$$upq = \frac{\eta_{pq}}{(m00)^{\gamma}}$$

where $\gamma = (\frac{p+q}{2} + 1)$

3. The first seven moment invariants are defined as:

$$M_1 = u_{20} + u_{02}$$

$$M_2 = (u_{20} + u_{02})^2 + 4u_{11}^2$$

$$M_3 = (u_{30} - 3u_{12})^2 + (3u_{21} - u_{03})^2$$

$$M_4 = (u_{30} + u_{12})^2 + (u_{21} + u_{03})^2$$

$$M_5 = (u_{30} - 3u_{12})(u_{30} + u_{12})[(u_{30} + u_{12})^2 - 3(u_{21} + u_{03})^2] +$$
$$(3u_{21} - u_{03})(u_{21} + u_{03})[3(u_{30} + u_{12})^2 - (u_{21} + u_{03})^2]$$

$$M_6 = (u_{20} + u_{02})[(u_{30} + u_{12})^2 - 3(u_{21} + u_{03})^2] +$$
$$4m_{11}(u_{30} + u_{12})(u_{30} + u_{12})$$

$$M_7 = (3u_{21} - u_{03})(u_{12} + u_{30})[(u_{30} + u_{12})^2 - 3(u_{21} + u_{03})^2] -$$
$$(u_{30} - 3u_{12})(u_{12} + u_{03})[3(u_{30} + u_{12})^2 - (u_{21} + u_{03})^2]$$

Appendix C

Selection of Color Space

Electro-magnetic radiation $F(\lambda)$ in the range of light ($\lambda \in [380nm..780nm]$) is perceived as colored light. The human eye color receptors divide the visible portion of the electro-magnetic spectrum into three bands: *Red, Green,* and *Blue*. For this reason, these three colors are referred to as the primary colors of human vision. By assigning each primary color receptor, $k \in r,g,b$, a response function $c_k(\lambda)$, visible light of any color $F(\lambda)$, can be expressed as a linear combination of the c_k's, as follows:

Normalizing c_k's to the reference white light $W(\lambda)$ such that:

$$W(\lambda) = c_r(\lambda) + c_g(\lambda) + c_b(\lambda)$$

$F(\lambda)$ can be expressed from (R,G,B) as:

$$F(\lambda) = Rc_r(\lambda) + Gc_g(\lambda) + Bc_b(\lambda)$$

Thus, any color can be represented as a linear combination of the three primary colors.

In this research work, we choose to use RGB format system to represent color of the video input. The RGB format is considered the most straightforward way to represent color using red, green and blue brightness values, scaled between 0 and 255.

It is possible to represent any color using a point in the color cube shown in Figure C.1. The origin of the RGB color space represents no brightness of any of the primary colors, i.e. *black*. Full brightness of all three colors appears as white. Three of the corners of the color cube are the primary color and the three others are *yellow*, *cyan*, and *magenta*. The diagonal going from the black corner to the white corner corresponds to the shades of gray and is called the *gray* line.

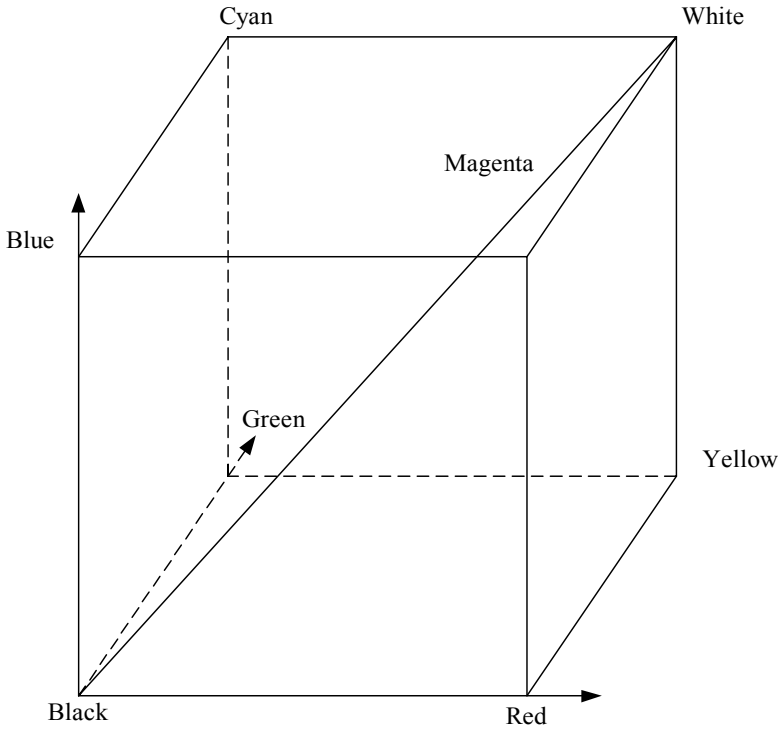


Figure C.1 The RGB Color Space Cube

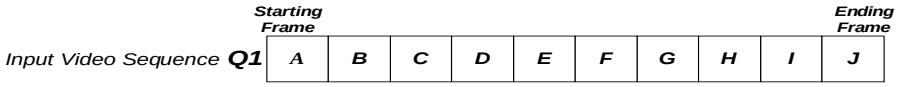
The RGB format system has been extensively used. Our television monitors use this system of overlaying Red, Green and Blue brightness values. The system can represent all colors that are visible to human eye and hence is a complete solution.

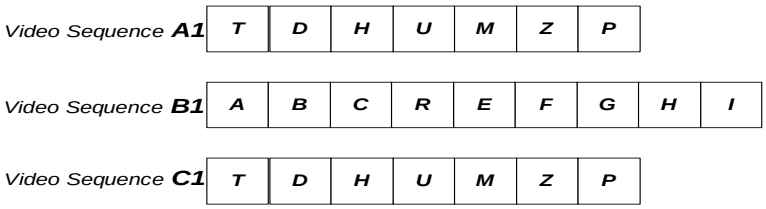
Appendix D

Illustration of Video Identification Process

In order to provide a better description and understanding of the video sequence identification technique described in this chapter, this section steps through a working example of the overall process of the identification technique:

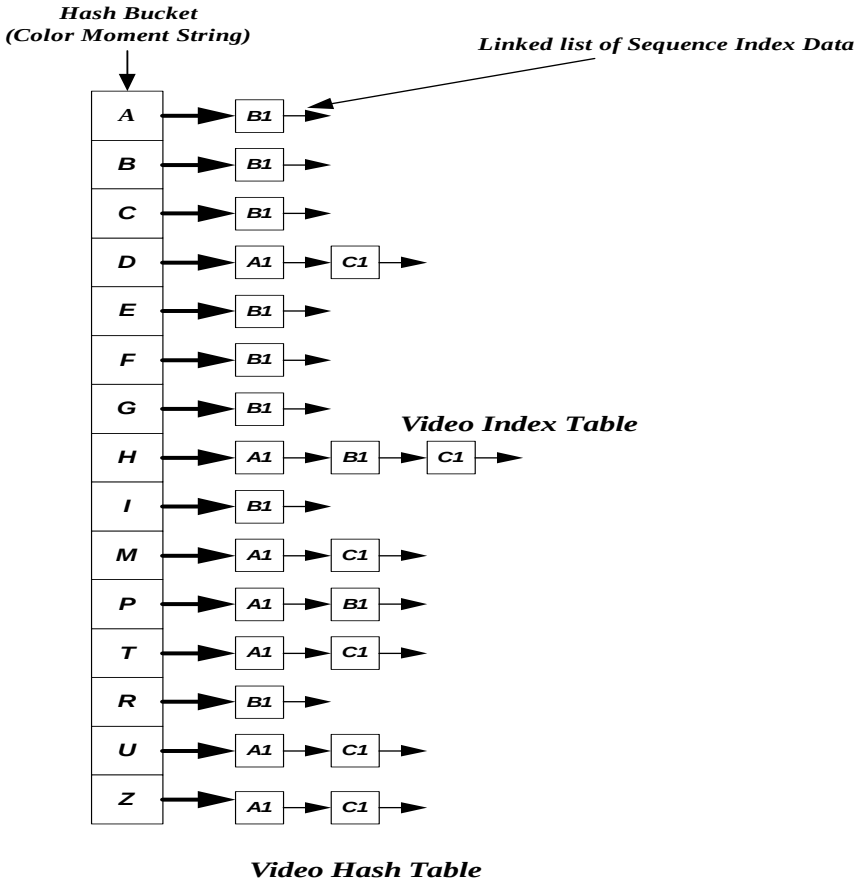
Given an input video sequence *Q1* with a size of 10 video frames, we want to compare this sequence to a video archive to find out if the input is a new video sequence or a repeat of a video sequence already in the archive. Assume that the current video archive contains 3 'old' video sequences: video sequences *A1, B1 and C1, and* each has a size of 9 video frames. In this example, assume that input video sequence *Q1* is a repeat of video sequence *B1* and that video sequence *C1* is a repeat of video sequence *A1*. The following drawing illustrates the content of these 4 video sequences with their respective color moment strings of each video frame.





Note :
Each block is a video frame with its color moment string represented by an alphabetic letter

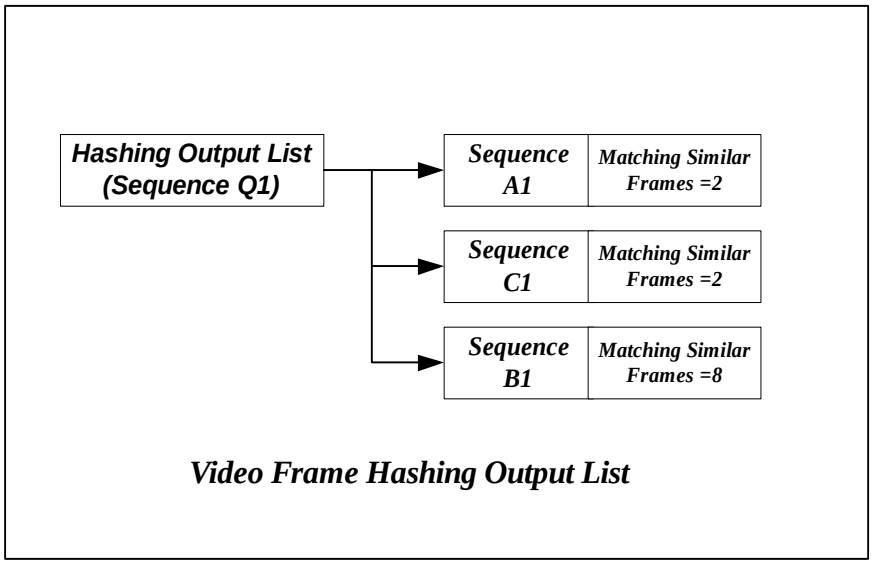
The content of the video frame hash table will look like this:



Video Hash Table

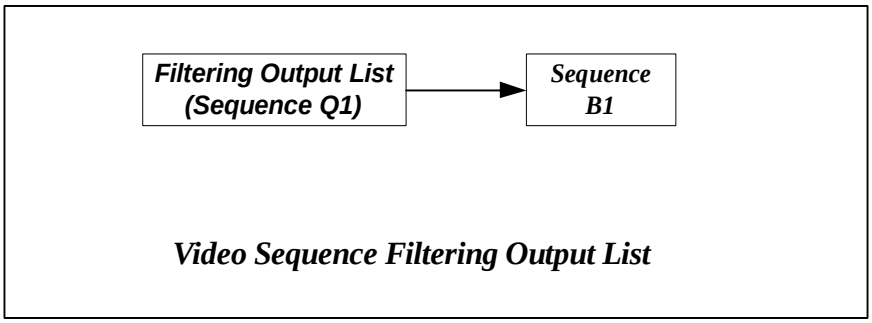
Step 1. Video Frame Hashing

The output after the video frame hashing process will have 4 potential similar sequences having at least one similar frame to the input video sequence *Q1*.



Step 2. Video Sequence Filtering

With an overlap threshold set to 30 and sequence size difference <10%, only sequence ***B1*** will pass the filtering and hence qualify as similar sequence for final step of the video identification process which is frame-by-frame video sequence comparison process



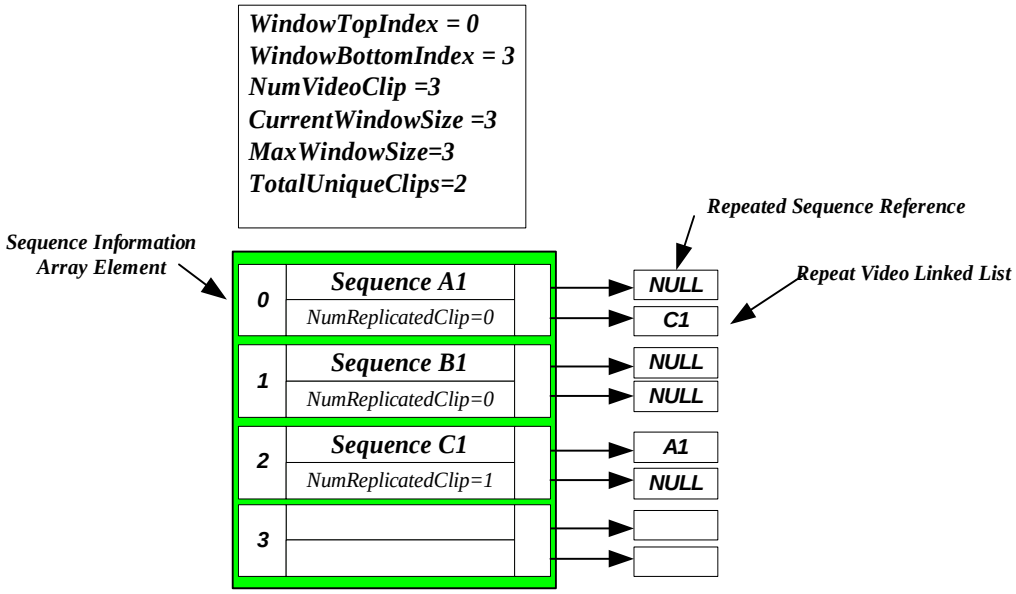
Step 3. Video Sequence Comparison

With a moment difference threshold set to 10.0, the system should identify video sequence *B1* as the only sequence of which Q1 could be a repeat. The meta-data of sequence *Q1* together with its matching result will be recorded into the video index table.

Step 4. Video Archiving and Tracking

Using the same input example scenario, the following example of video index table update provide a better understanding and graphical description of the whole process which include insertion of the input video sequence ***Q1*** and a deletion of the expired sequence ***A1.*** Assume that our identification system has a sliding window size of 3 video sequences. That means that the video archive is allowed to grow to a maximum size of 3 video sequences.

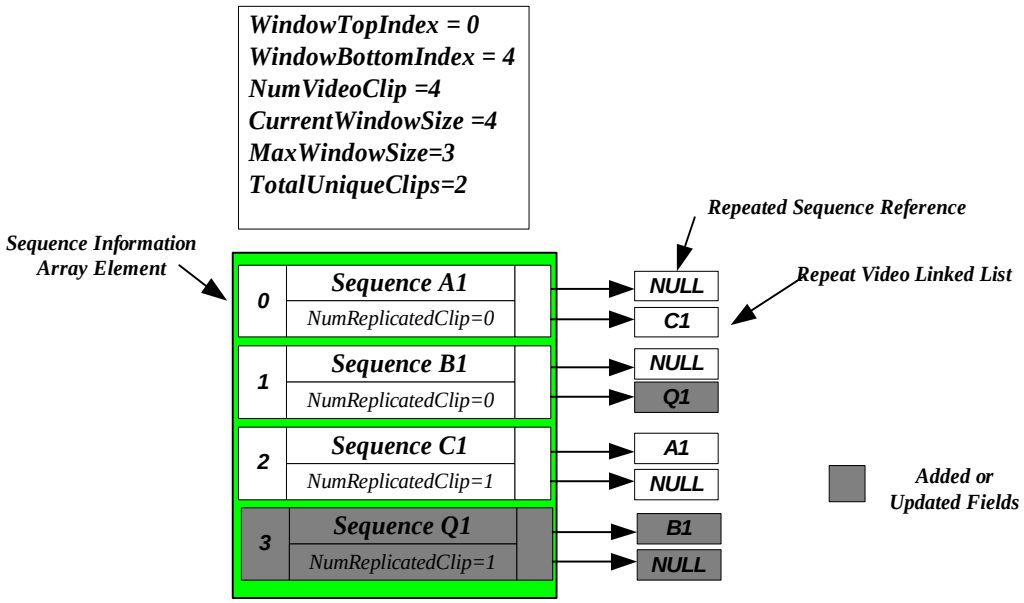
I. The current video index table before the input video sequence insertion



Video Index Table

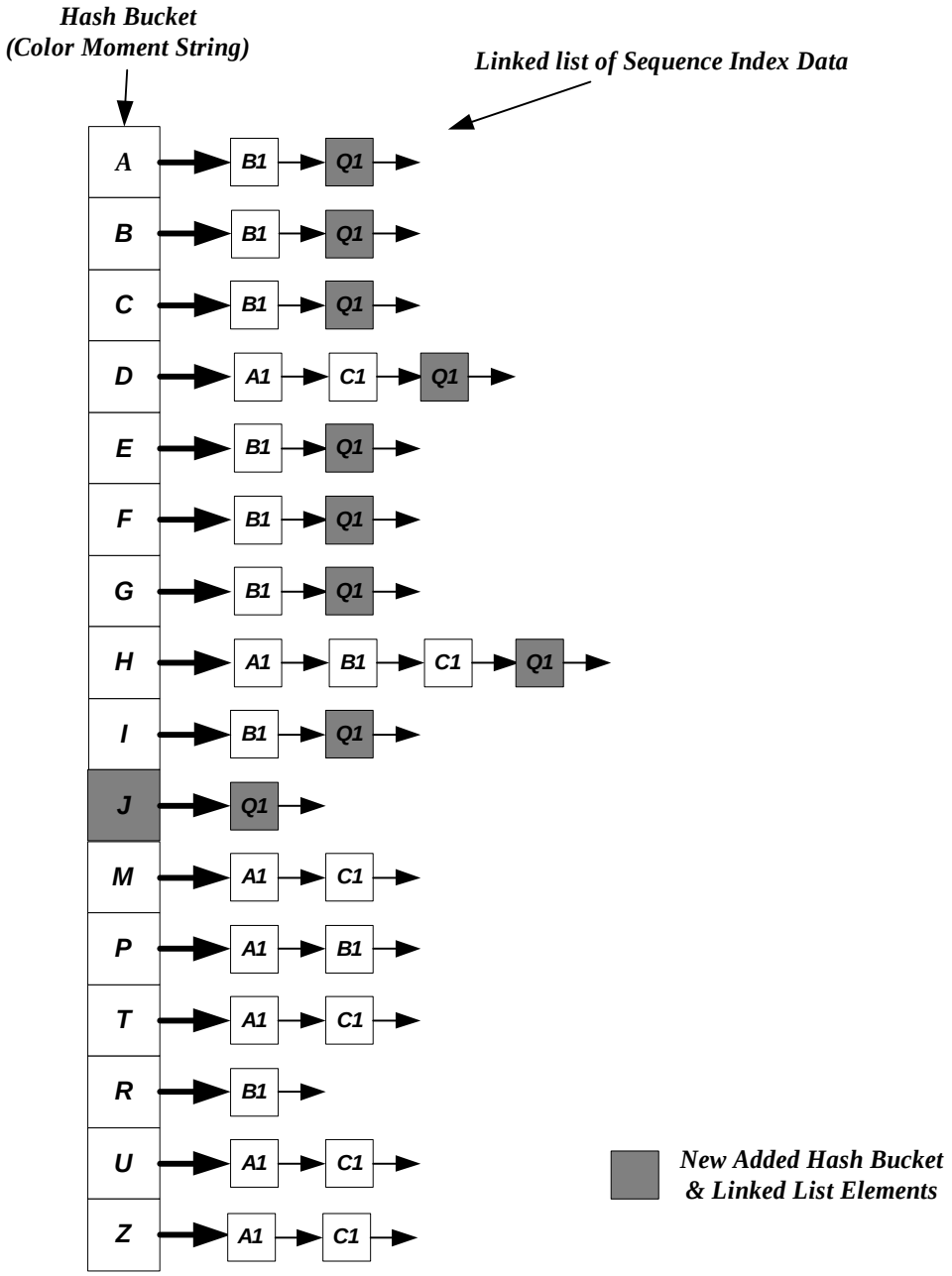
II. Insertion of the Input Video Sequence Q1 into the Video Index Table

Video sequence Q1 will be inserted into *Array Element 3* as repeat sequence. Increment WindowBottomIndex and add *Sequence Q1* into *Sequence B1's Repeat Video Linked List*. Now the total sequences in the video archive (NumVideoClip) is 4 and the number of unique sequences detected (TotalUniqueClips) remains at 2.



Video Index Table after Insertion

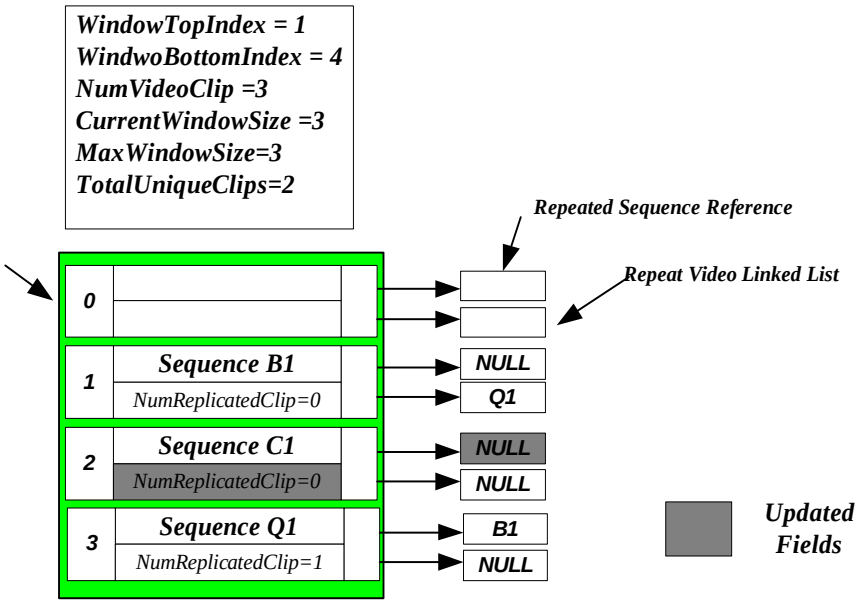
III. Insertion of Color Moment String of Sequence Q1 into the hash table



Video Hash Table after Insertion

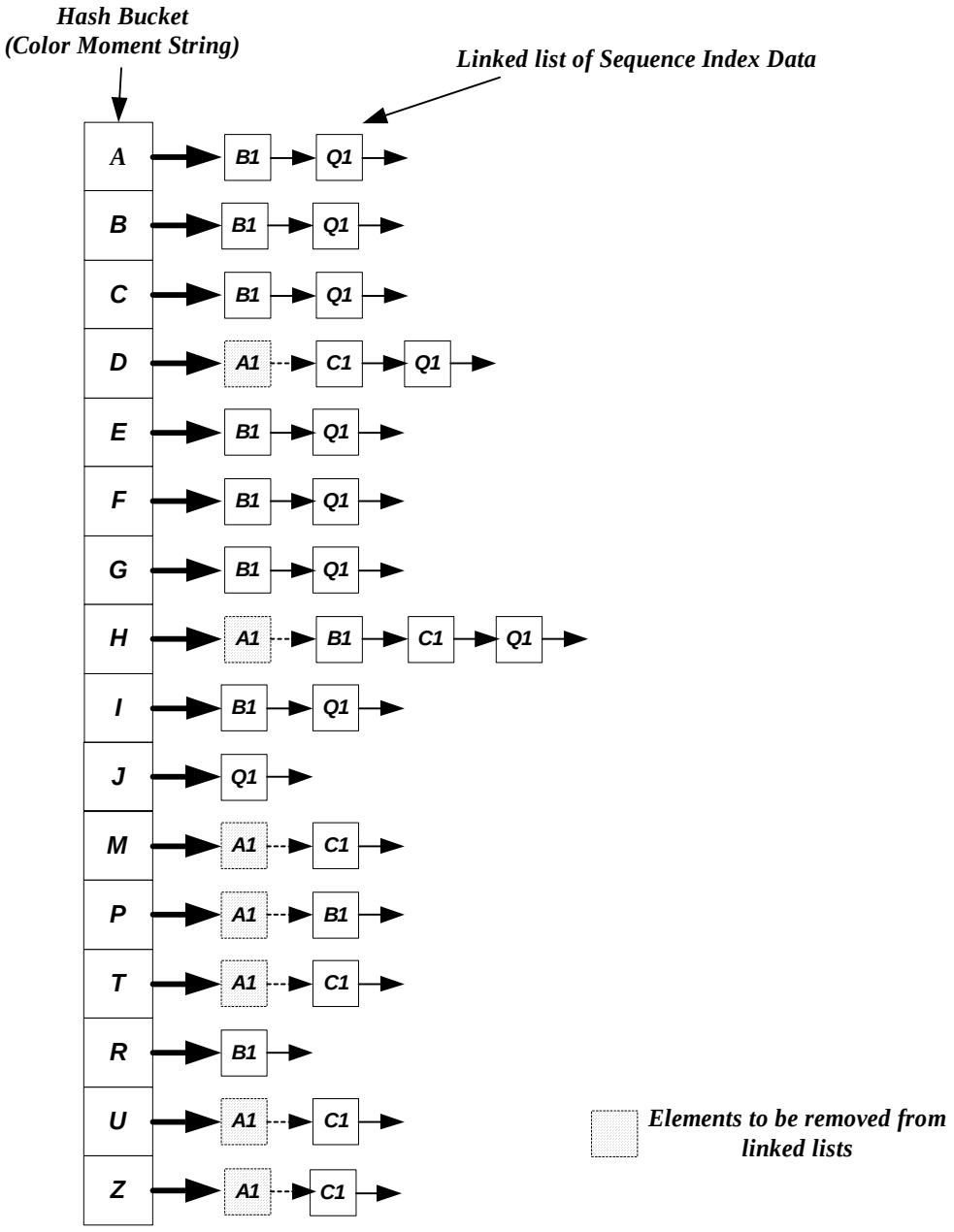
IV. Deletion of Expired Video Sequence from the Video Index Table

Since there is a size overflow of the video archive, the expired sequence (*sequence A1)* stored in *Array Element 0* needs to be removed. With the removal of *sequence A1*, *sequence C1* will now become "new" and hence its Repeated Sequence Reference field is set to NULL. The NumReplicatedClip field is set to 0 because it is a new sequence after the deletion of the expired sequence.



Video Index Table
After Expired Sequence Deletion

V. Deletion of Color Moment Strings of the Expired Video Sequence from the Video Hashing Table



Video Hash Table Deletion

Appendix E

Measuring Recall and Precision

Assume we start the video identification process with zero video archive. There are a total of 7 new input video sequences, namely *S1.1, S1.2, S1.3, S1.4, S2.1, S2.2*, and *S2.3*. Sequences S1.2, S1.3, and S1.4 are repeats of sequence S1.1. Also, sequences S2.2 and S2.3 are repeats of S2.1. The results for each input video sequence are as such:

1. Sequence S1.1:

   It is detected as a new sequence.

   Since it **is** the first occurrence of this sequence, the system is credited with a true negative.

2. Sequence S1.2

   It was detected as a repeat of sequence S1.1.

   S1.2 is a repeat of S1.1 and hence the system scores a true positive.

3. Sequence S1.3

   It was detected as a new sequence.

   S1.3 is a repeat of S1.1 and S1.2. This is scored as two false negatives since S1.3 is incorrectly identified as new when in fact there are two matching sequences in the archive.

4. Sequence S1.4

   It was detected as a repeat sequence to Sequence S1.1, S1.2 and S1.3.

S1.4 is a repeat of sequences S1.1, S1.2 and S1.3. Thus, the result is scored as three true positives.

5. Sequence S2.1

It was detected as a new sequence.

S2.1 is the first occurrence of a new sequence; therefore, the result scores one true negative.

6. Sequence S2.2

It was detected as repeat sequence for S2.1.

Since this is correct, the result scored as one true positive.

7. Sequence S2.3.

It was detected as a repeat of sequences S2.1 and S1.4.

Sequence S2.3 is only a repeat of S2.1. Therefore, the result is scored as a true positive on sequence S2.1 and a false positive on sequence S1.4.

The following table records results of the identification of these 7 input sequences:

Input Sequence	Total Detected Repeated Sequences	Correctly Detected Repeated Sequences	True Repeated Sequence	True Positive	True Negative	False Positve	False Negative
S1.1	0	0	0	0	1	0	0
S1.2	1	1	1	1	0	0	0
S1.3	0	0	2	0	0	0	2
S1.4	3	3	3	3	0	0	0
S2.1	0	0	0	0	1	0	0
S2.2	1	1	1	1	0	0	0
S2.3	2	1	2	1	0	1	0
			TOTAL	***6***	***2***	***1***	***2***

With a video archive size of 7 video sequences, the recall and precision can be calculated as below:

Recall = true positives / (true positives + false negatives) = 6 / (6+2) = 0.75

Precision = true positives / (true positives + false positives) = 6 / (6+1) = 0.85