2005

Query engine of novelty in video streams

James Kang

A Query Engine of Novelty in Video Streams

Master's Thesis

James M. Kang
Department of Computer Science
Rochester Institute of Technology
jmk4644@cs.rit.edu

Thesis Committee:
Dr. Ankur M. Teredesai, Chair
Dr. Vladimir Misic, Reader
Dr. Roger S. Gaborski, Observer

Abstract

Prior research on novelty detection has primarily focused on algorithms to "detect" novelty for a given application domain. Effective storage, indexing and retrieval of novel events (beyond detection) are largely ignored as a problem in itself. In light of the recent advances in counter-terrorism efforts and link discovery initiatives, the need for effective data management of novel events assumes apparent importance.

Automatically detecting novel events in video data streams is an extremely challenging task. The aim of this thesis is to provide evidence to the fact that the notion of novelty in video as perceived by a human is extremely subjective and therefore algorithmically ill-defined. Though it comes as no surprise that current machine-based parametric learning systems to accurately mimic human novelty perception are far from perfect such systems have recently been very successful in exhaustively capturing novelty in video once the novelty function is well-defined by a human expert. So, how truly effective are these machine based novelty detection systems as compared to human novelty detection? In this paper we outline an experimental evaluation of the human vs machine based novelty systems in terms of qualitative performance. We then quantify this evaluation using a variety of metrics based on location of novel events, number of novel events found in the video, etc. We begin by describing a machine-based system for detecting novel events in video data streams. We then discuss the issues of designing an indexing-strategy or 'Manga' (comic-book representation is termed as 'manga' in Japanese) to effectively determine the 'most-representative' novel frames for a video sequence. We then evaluate the performance of machine-based novelty detection system against human novelty detection and present the results. The distance metrics we suggest for novelty comparison may eventually aide a variety of end-users to effectively drive the indexing, retrieval and analysis of large video databases. It should also be noted that the techniques we describe in this paper are based on low-level features extracted from video such as color, intensity and focus of attention. The video processing component does not include any semantic processing such as object detection in video for this framework. We conjecture that such advances, though beyond the scope of this particular paper, would undoubtedly benefit the machine-based novelty detection systems and experimentally validate this. We believe that developing a novelty detection system

that works in conjunction with the human expert will lead to a more user-centered data mining approach for such domains.

JPEG 2000 is a new method of compressing images better than other image formats such as JPEG, GIF, PNG, etc. The main reason this format is in need for investigation is it allows metadata to be embedded within the image itself. The types of data can essentially be anything such as text, audio, video, images, etc. Currently image annotations are stored and collected side by side. Even though this method is very common, it brings up a lot of risks and flaws. Imagine if medical images were annotated by doctors to describe a tumor within the brain, then suddenly some of the annotations are lost. Without these annotations, the images itself would be useless. By embedding these annotations within the image will guarentee that the description and the image will never be seperated. The metadata embedded within the image has no influence to the image iteself.

In this thesis we initially develop a metric to index novelty by comparing it to traditional indexing techniques and to human perception. In the second phase of this thesis, we will investigate the new emerging technology of JPEG 2000 and show that novelty stored in this format will outperform traditional image structures. One of the contributions this thesis is making is to develop metrics to measure the performance and quality between the query results of JPEG 2000 and traditional image formats. Since JPEG 2000 is a new technology, there are no existing metrics to measure this type of performance with traditional images.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Large amounts of data itself are very hard to analyze and understand. One of many areas that are emerging is the area of Novelty Detection. This concept refers to the occurrence of something new, different, or interesting from what has been previously seen. For much of the research conducted on Novelty Detection, the amount of data analyzed is quite large. There has been work done on finding correct novel events from a set of data. However, reviewing these events by analysts' still takes time and is quite complicated.

A proposed solution to review these novel events is to index these events in such a way that is more efficient and easier to examine. For any video sequence, novel frames are extracted from the image. These novel frames represent new events that occur within the video. Since there can be an infinite number of novel frames, a modified clustering approach has been developed to organize this data (Section 3). Once represented frames are found from each cluster, these will be used to store within an index. Instead of finding the features of the whole image, novelty regions of the image will be extracted and its features will be stored within the image in a JPEG 2000 format. By utilizing JPEG 2000's capabilities, this thesis will show that the retrieval time of the query search engine will be far more efficient that the query time of traditional image formats. New metrics will need to be developed to measure the performance in terms of speed an quality between JPEG 2000 and traditional image formats.

The next chapter of this thesis is an extensive survey regarding to Novelty Detection and its related work. Chapter three explains the types of novelty detection systems we used, how they were compared, the novel extraction process, clustering and our results. The fourth chapter dealt with an extensive survey and implementation of various multimedia indexing schemes. The fifth chapter describes the technology of JPEG 2000 and a modified application to annotate specific regions within an image. The implementation details for the novel query engine is discussed in chapter six. Finally, an extensive analysis of our results and opportunities for future work is discussed in chapter seven.

1.1 System Overview

Industries all over the world produce hours and hours of surveillance videos every day to protect their own well-being. But its often very difficult and tedious to sift through all of these videos to find interesting occurances. Even if an interesting event is found the liklihood of finding similar events in large database of video frames is "non-trivial."

A large body of work has been attempted to solve this problem by identifying events in a video. Xie et al. have shown events can be described as genres in video streams [45]. Pan and Faloutsos categorize events from news videos into two classifications of news and commercials using their VideoCube system [32].

Segmenting the video has been very popular to discover events. Zhang et al use camera transitions to segment sport related videos, similar to chapters in a book [49]. The system ClassMiner [51] designed by Zhu et al identifies segments based on the background in video. Scenes are found within each segment by the similarity in coninuous frames and events detected by sound such as cheering or music. Wang et al use human motion to discover events such as walking [43]. Surveillance videos can be segmented based on using motion and background information as described by Oh et al [30]. Segmented events are converted to textual descriptions for fast retrieval by Windhouwer et al [44].

However, video summarization is meaningless since the events may not be meaningful to the viewer. A filter is needed to identify only interesting events, also coined as Novelty Detection. This concept refers to the occurrence of something new, different, or interesting from what has been previously seen. Events can be learned by general patterns of activity within a scene [37]. Diehl et al has been able to classify novelty based on previous events using motion [14]. Crook et al implemented a neural network to habituate to the surrounding elements and classify stimulus patterns as familiar or novel.

Habituation is a cognitive effect by which the biological system ceases to respond after repeated presentations of the same stimulus [35]. Marsland et al applied habituation to computational modeling through a motion camera on robots [33]. Habituation has been successfully applied to automatically detect interesting events in video data as described by the VENUS (Video Exploitation and Novelty Understanding in Scenes) system [16].

In order to provide a meaningful video summary especially in the surveillance video domain, a novelty filter must be used and represented effectively. In this paper, we have chosen to use VENUS to provide our data set and use our own data mining techniques to cluster, index, store, and query for similar events. Our proposed framework is shown in Figure 1.1 which is summarized as follows.

- In phase one of the framework, as the video frames are processed, the features are

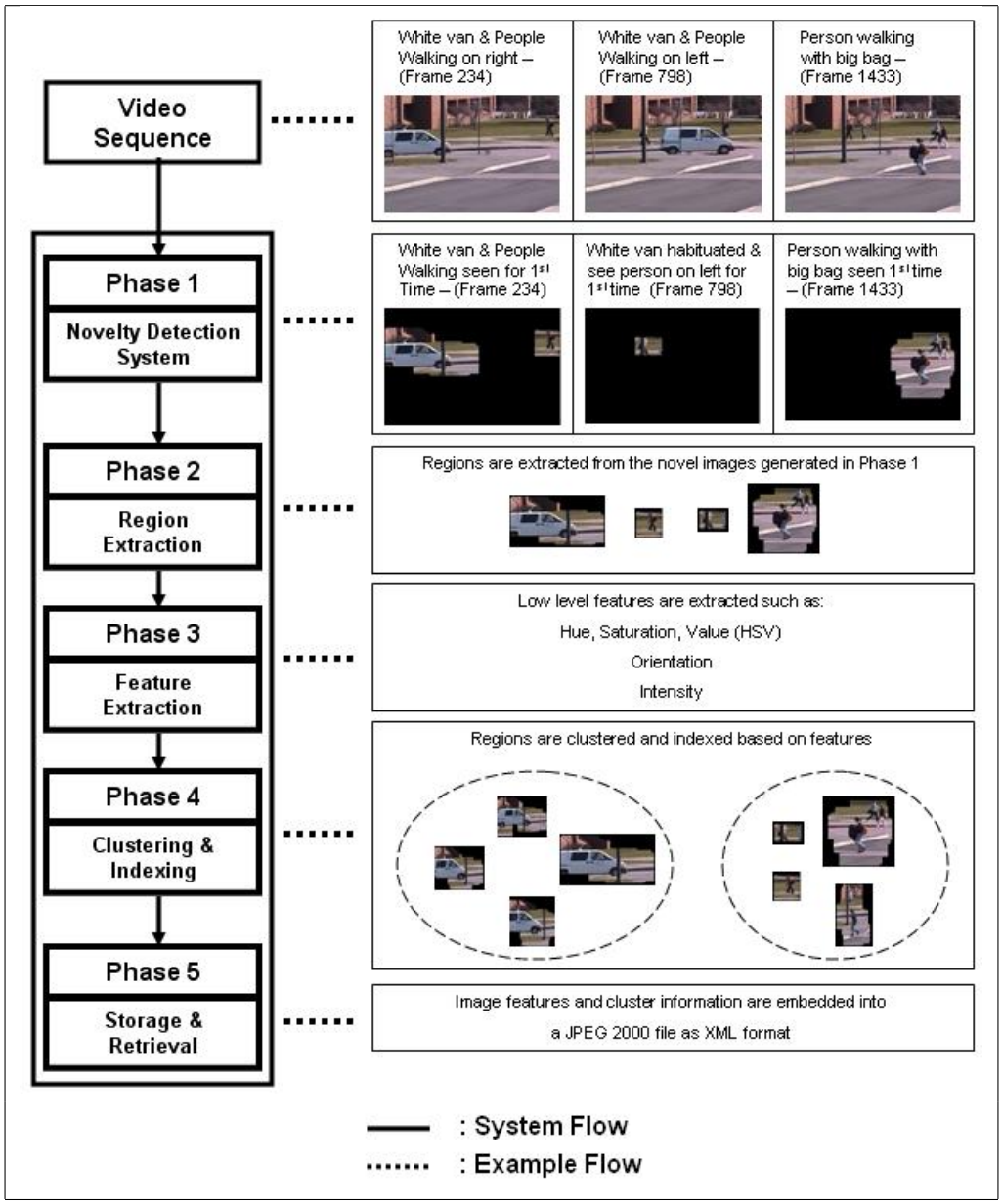


Figure 1.1: Query Engine of Novelty Framework and Example

extracted based on motion, still, and color over a period of time. In the example, the white van is seen novel since initially anything seen for the first time is novel. After a time period, the system loses attention toward the white van and no longer considers the van as to be novel. This process is described in detail in Section 2.

- Since multiple events can occur in a single frame (Phase 2 in Figure 1.1), regions need to be extracted to identify a single event. The features found in each region will be more meaningful than an image containing several events.

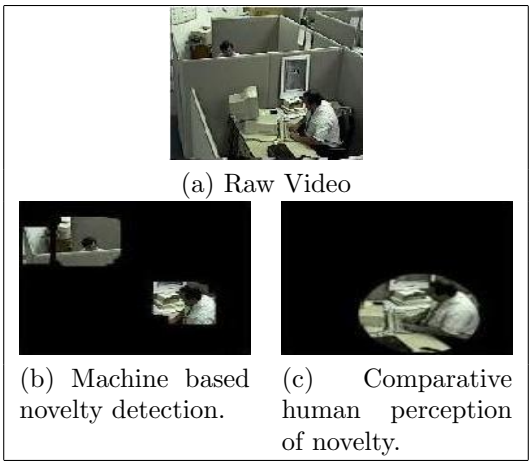
- In our framework, we are extracting low level features to describe each event in Phase 3. We use Hue, Saturation, Value (HSV);Orientation and Intensity.

- As an event is habituated, multiple images are created for a single event. In the example for Phase 4 in Figure 1.1, several regions for the white van was found before this event was habituated are clustered together. We index this information using a R-tree approach. We then compare the R-tree approach with several state-of-the-art spatial indexing techniques and analyze the utility of each approach.

- Finally, as shown in phase 5 of Figure 1.1, we utilize the technology of JPEG 2000 to store the image, features, and cluster information together in a single file. We developed a query engine to retrieve similar interesting events on top of this framework.

1.2 Extracting Novelty Using Various Novelty Detection Systems

Novelty detection in video data streams is a challenging task. We as humans can process video data and can pin-point the interesting events we observe. Pattern recognition or multimedia data mining systems that provide similar functionality can greatly aide the human analyst in several domains including surveillance, marketing, etc. Such novelty detection systems are gaining importance not only due to the complexity of the task, but also due to the large volume of video data becoming available in a digital format. The goal of this paper is to determine if such "machine" based novelty detection systems are up-to-par to what the human analyst perceives as interesting in video data streams (Figure 1.2). Analysis techniques and corresponding distance metrics such as the ones proposed in this paper provide significant insight into the way a human visual system perceives novelty and helps us understand the shortcomings and strengths of an automated novelty detection system.



(a) Raw Video

(b) Machine based novelty detection.

(c) Comparative human perception of novelty.

Figure 1.2: Comparing human vs. machine based novelty detection. The top image is the raw video frame, on the left is what the human subject thought was novel, and on the right is what the machine based novelty detection algorithm detected as novel. These are representative images of novel frames based on novelty clustering we describe in this paper. Notice that (a) the novelty locations are very similar; (b) the machine based system is more exhaustive and detects more events as novel; (c) comparatively, the human subject found only one event as novel.

1.2.1 Comparing Human Perception and Machine-based Novelty Detection

A more user-centered KDD approach is gaining in importance in recent years [18]. Multimedia data mining, specifically mining video data streams is one of key areas that can benefit from a more user-centered data mining approach due to its very nature of being extremely sensitive to human perception. Statistics suggest that an average American teenager watches approximately 25,000 hours of television in their teens [48]. A security analyst responsible for detecting suspicious behavior at any major airport simultaneously monitors approximately 20 live video feeds. Hence, identifying novel events in large video data streams has extensive uses in commercial as well as law-enforcement applications.

Effective methods to summarize large video collections based on novel or important aspects of the video are being explored by a wide range of industries. Recently, the New York Times reported an article mentioning several video conferencing and video archiving industries being interested in ways to capture important sections of meetings and make an outline of each meeting available for future reference. Likewise, security/surveillance based industries are looking for ways to detect important events in huge streams of unimportant video. The task of going through the vast footage of video to determine and then summarize the interesting things is not trivial for a human analyst. Recently, Gaborski et al developed a machine based novelty detection system to scene understanding from video data streams [16]. They compared their learning system with other state-of-the-practice video novelty detection systems and reported the results. In this paper we use their VENUS framework as the

machine based novelty detection system. The human novelty detection system in this paper refers to the formulation of the human subject studies we conducted. We will use the Eye-Tracker device as a data collection tool to detect the human attention regions based on eye movements of a subject while watching a video playing on a TV monitor. Details of how to derive a formulation of novelty from these attention areas and the subsequent comparison with the machine based novelty detection system VENUS are also described in detail in the paper.

1.3 Indexing Novelty

In order to effectively represent novelty from a video stream, an efficent indexing scheme is needed. In this thesis, we initially studied indexing fundamentals by implementing a database containing novelty using a hash table, B+tree, and a R-tree. Following this implementation, we began to survey and implement state-of-the-art R-tree varied indexing algorithms. We compared the performance of these indexing schemes using our own metrics against the original R-tree implementation.

1.3.1 Indexing Novel Frames Using Spanning Blocks

In this project, we made use of a novel database made of two relations Still and Motion which consisted of features taken from novel images in a video stream. The Still table was used to represent the attributes of a still image, while the Motion table was used to represent a frame made up of seven images.

The tables were stored in the form of two flat files. We started the project by first directly indexing the flat files using three different indexing schemes. The indexing schemes we used where HashMap, B+ Tree and R Tree. We modified a SQL parsing package we found on the internet (ZQL) to work with our indexes such that we were able to pass different queries on the flat files, as well as insert and delete from our database.

In the second part of the project we created blocks of specified size to store the records and indexed them instead of indexing the flat files directly. We did this by first implementing the database using fixed blocks which had to contain the complete record and had to be bigger then the record size. We later modified our design of blocks to spanning blocks which could be of any size and allowed records to span across them.

We also implemented cost estimate calculation for each query in our parser and compared it with actual cost values by executing the query in terms of total I/O bytes read/written from the blocks or tuples.

1.3.2 Comparative Analysis of Various Multimedia Indexing Algorithms

We surveyed approximately twenty different indexing schemes and we narrowed it down to four algorithms that we could investigate and would be best for our data set. The algorithms we studied are the PR-tree designed by Arge et al [7], a Singular Value Decomposition based index by Li et al [22], HDoV-tree by Shou et al [34], and the P-tree by Crainiceanu et al [11]. These indexing schemes are based on a R-tree [17] where its intended use is for spatial data.

The PR-tree is based on minimizing the cost of querying by storing extreme values within the index. A main disadvantage of a R-tree is when querying for a region containing both minimum and maximum values. This causes a region that covers the whole R-tree which is very expensive. Arge et al have designed a two step indexing algorithm. Initially they created a Pseudo PR-tree which efficiently stored the extreme values at each node. Then they would use this index to convert it into a PR-tree. The design is similar to a R-tree but they minimize the size of the bounding box by knowing these extreme values beforehand.

Singular Value Decomposition (SVD) is a process of decomposing a large multivariate m x n matrix where $m \geq n$ to a small matrix while maintaining the same variability between each row. The decomposed values are stored in a suffix like tree. In our data set, we also have a large matrix consisting of eight features as m and n images found by a novelty detection system. We have implemented SVD along with a suffix like tree and shown that the performance has improved over a R-tree.

An interesting method of viewing data is using a HDoV-tree. This tree is initially intended to view 3D objects based on two principles. The first principle is called Level of Detail (LoD). For example, imagine if you were standing in front of a set of building such as a school campus. The buildings closer to you will have a higher level of detail since you can see it more clearer. Where farther building will have a lower level of detail. The second principle is called Degree of Visibility (DoV) which is based on your point of view in our example. Smaller buildings behind larger buildings will not be seen and thus not important to your point of view. Obviously our system does not deal with 3D objects and uses image, but we have implemented creative interpretations of these principles and have shown positive results.

Our last indexing scheme we reviewed is based on peer-2-peer (P2P) networks called the P-tree. We were interested in this index since P2P networks have been growing in popularity for the past couple of years. This structure is very important to our data set which allows for scalability of our system. We have used the designs of a P-tree and applied it to our data set.

1.4 Archiving Novelty Indicies Using the JPEG 2000 Standard

JPEG 2000 has the ability to display the image without using all of the image data. For example, let's say we have aerial images taken from satelite stored at a server in NASA. These images can range from a couple megabytes to gigabytes of information. If a user was located on the other side of the world and wanted to view these image, the load time would a really long time. Since JPEG, GIF, and other image formats cannot stream the information to the user. In order for the user to view the full image, he/she needs all of the image data. In JPEG 2000, the data can be seperated into pieces based on resolution. Even with a dial up connection, the server could send the image at a very low resolution which can be considerably lower than the original size and be displayed to the user a lot faster. The user can then ask the server for more image information and the server will improve the image quality. This allows the image to be streamed to the user, similar to how videos are streamed. Where parts of the video can be sent over the internet.

In general, image annotations are based on the whole image itself. These annotations can either be short and nebulous or long and specific. The problems with annotations today is that its too broad and doesn't really describe what is going on within the image. An image search is based on the annotations for the whole image. Without knowing the details of an image based on one annotation, it is fairly hard to find the image your looking for. In JPEG 2000, it allows the ability to select a region from an image and store an annotation for that region. An infinite amount of regions can be selected and annotated without affecting the image itself. This allows a far more efficient image search where more information of an image can be extracted. Also, the seperate regions of an image can be extracted from the image and sent back to the user. This is very benificial when the image is quite large, and the query matches to only part of the image. This part of the image can be sent back to the user with different resolutions.

Before we could begin to incorporate JPEG 2000 into our query engine, we had to understand and experiment with this technology. Thus in this thesis, a tool is created to select a region of an image and annotate the region. The annotated information is stored within the image in XML format. This format allows us to read the data more efficiently. Once the annotated image is created, an upload feature was developed to automatically extract the information from the image and store it within a database. This allows a user to search through the images based on the annotations and return the equivilent images. Then we applied this technology to our query engine by having a reliable storage utility for both the image and its information.

Chapter 2

Related Work

A survey was performed on different types of Novelty Detections, which include Applications, Clustering/Indexing, Sub-graph Isomorphism, and Time Series. This was conducted to understand what types of work has previously been performed in this area as well as to formulate and discuss thesis topics in this field.

2.1 Applications

The survey conducted for these applications of Novelty Detection center around a single goal, which is to determine new and interesting events based on previous events. There were numerous types of applications dealing with Novelty Detection, but there were a couple of applications that dealt with this topic. Mammography [38] and Robots [33][12] use images as input for these systems. Video Streams were also used within a Video Surveillance application [14].

2.1.1 Image Novelty Detection

Even though this topic is dealing with video streams, Image detection is an important step to analyze video and its Novel events. The applications discuss here use low level features which are very similar to VENUS. Detecting Breast Cancer efficiently has always been an ongoing problem. According to Tarassenko [38], there are about 26,000 new cases in the United Kingdom each year. On average, there needs to be at least two analysts to review an x-ray image to determine if there are any cancerous areas within the breast. This paper doesn't attempt to develop an application to identify if a person has breast cancer or not. It is merely a tool for analysts to focus on areas with larger mass regions in certain parts of the images. This could allow an expert to look at areas that are most important.

Novelty Detection is performed by using features of shape, texture, boundary (edges), and contour. These features will then distinguish between normality and abnormality. Based on the feature vector, a density function is used to find a certain value. If this value is below a

pre-defined threshold, then it is considered to be novel. Humans and animals have a natural instinct about detecting something new or out of the ordinary comes into their surroundings. These two papers used this observation and developed their novel applications using robots. In Crook, Marsland, and Hayes's paper [12] it used images taken from the robot, and in Marsland, Nehmzow, and Shipiro's paper [33] they used both Image and sonar readings taken from the robot.

The features taken within Crook, Marsland, and Hayes's paper [12] were similar to the features taken in the VENUS [16] project. They both used features of color, intensity, and orientation. However, in Marsland, Nehmzow, and Shipiro's paper [33] they did not use the same features, but they used the same concept of VENUS with the idea of Habituation. This idea is that as more repetitive events take place, the less novel the event is.

In Marsland, Nehmzow, and Shipiro's paper [33], they use the idea of neurons and a synapse, similar to a Neural network where the events are read in from each neuron and the synapse determines if it is novel.

2.1.2 Video Novelty Detection

Video Surveillance has been a major concern especially since the September 11th attacks. Diehl and Hampshire [14] examine the novelty that occurs within a video. It also attempts to classify an object based on previously labeled objects. Initially, each image is classified with a label and then with a sequence of images.

There are a couple of main differences between this project and VENUS' project. A motion detected camera is used. So automatically they can assume that all of the video can be considered as motion. Where in VENUS, color and still can also be considered to be novel. There also doesn't seem to be a feature set, or it wasn't clearly stated within the paper. It seems that they are comparing the images with a pre-classified set of images, and that's how they find events within a video.

2.1.3 Other Applications

Further applications were reviewed in this survey to gain some general knowledge in the area which were Sentence Level Detection [4], Jet Engines [28], Topic Detection [47], and Mass Spectral Data [40]. These were reviewed to only understand different types of technologies that exist today and not to improve or investigate further on these applications.

TREC 2002 was held in Gaithersberg, Maryland to discuss applications of text retrieval. There were numerous topics within this field and this paper focused on retrieving relevant sentences that occurred within one or many documents. Allen, Wade, and Bolivar [4] developed their system based on a Vector Space Model. This model can be classified into three parts. The first part is where the document is indexed on the sentences. The sentences

are then weighted based on their relevance to each other. Finally, these are then ranked together.

Novelty was found by using features of word counts and relevance. Distance was measured on these features. There were Language Models that were set up, which assigned degrees of novelty in each occurrence. When a jet engine is created, it is hard to tell whether the engine is fully working unless it is tested exhaustively [28]. Instead, a novel approach can be taken by comparing the vibrations the engine has created based on vibrations taken from fully working jet engines. The vibrations produce a value of high pressure (HP) which could then be used to compare new and old engines.

Novelty is found based on previous data. Training is needed by having the system look at normal working engines. Then, when a new engine is created, it can be compared to previous data and determine if the engine is abnormal or not. For any set of documents, Yang, Zhang, Carbonell, and Jin [47] attempts to group or cluster the documents in forms of topics. They briefly mention that this algorithm is an improvement over the First Story Detection (FSD) algorithm. Basically, FSD classifies the documents in a Brute Force fashion. It checks all the previous topics with the new topic. For each topic, different types of events can occur. Events are generally more specific, such as "East End Festival", whereas topics are more general , such as "Street Festivals".

They base their algorithm on using an FSD for each topic, rather than using all the documents in one FSD. Like Query Expansion, they use a list of stop words so they can ignore any redundant words. The list of features they use for Novelty Detection are named entities. Some examples of named entities are time, place, money, etc. These features are then used to compare with other topics to determine novelty.

Mass Spectrometry is the detection of important characteristics of the molecular structure of a certain compound. Tong and Svetnik [40] discusses the difficulty on finding certain structures within fungi. Fungi is considered to be very hard to grow and unstable. An application of novelty was to be developed to identify any new characteristics. This could expand into drug discovery as an application.

The basic algorithm in this paper uses a Support Vector Machine (SVM) SVM's can naturally create outliers within its algorithm. These outliers can then be considered to be novel. It was found in this paper that it is an improvement over using Temporal Sequences [14].

2.2 Clustering/Indexing of Novelty

Video surveillance, scene understanding and consequently novelty detection in video as a counter-terrorism measure have emerged as a major technical challenge in recent years, especially since the September 11th attacks. There are several notable efforts that perform

event detection in video. We outline a few of these relevant efforts that focus more on 'novelty detection' beyond merely event detection. Typically, novelty detection in video is performed using features such as shape, texture, boundary (edges), and contour. These features are then used to distinguish between normal and abnormal events.

Diehl and Hampshire [14] examine the concept of novelty that occurs within a video. They attempt to classify an object based on previously labeled objects. Initially, each video frame is treated as an image, classified with a object as its class label, and then grouped with a sequence of images. Object detection in images is non-trivial and imprecise, it is difficult to train such a system and use it for novelty detection. VENUS [16] takes a clustering of data streams approach to novelty detection. It not only considers motion features but also includes color and orientation to determine novelty in video frames. It uses a mixture of gaussians to model the aspect of novelty clusters and then groups events based on the existing gaussians. The distinction between a regular event and a novel event is based on a 'habituation' function they describe. Living beings have a natural tendency to detect and differentiate something as new or out of the ordinary within their surroundings. Subsequently, Marsland et al [33] use both image and sonar readings to detect novelty. Habituation is an effect by which a system ceases to respond after repeated presentations of the same stimulus [35]. Work by Paul Crook [12] implements a neural network to habituate to the surrounding elements and classifies stimulus patterns as familiar or novel. Computational modeling of habituation has been proposed by Wang [6]. The basic idea behind 'habituation' is that as an event gets more repetitive, the less 'novel' that event becomes, since the system 'habituates' to that event's input.

Burl et al suggest extracting features from a video stream using a background subtraction algorithm [9]. Their system assumes that the background will be the same throughout the video and any changes to the foreground causes a 'novel' event. This assumption is rather simplistic and inapplicable if the background changes. They also attempt to track objects in motion throughout the video. When two objects cross over each other, the tracking of the initial object is lost, leading to an imprecise detection of novelty.

Even though this paper deals with video data streams, novelty detection in raw image data is an important area closely related to novelty detection in video. For example, detecting breast cancer effectively in MRI/X-Ray images has received significant attention in the medical imaging community [38]. According to Tarassenko [38], there are about 26,000 new cases of this disease in the United Kingdom each year. On average, at least two analysts need to independently review an x-ray image to determine if there are any cancerous areas detectable. Thus, effective user-centric tools to detect novelty in medical images will have widespread usage.

A comprehensive survey for novelty detection using statistical methods is provided by

Markou and Singh [26]. With the focus on surveillance and monitoring applications increasing, there is considerable emphasis on novelty detection within the data mining community. Yamanishi and Takeuchi [46] discuss novelty detection for non-stationary (adaptive) data while incorporating the effect of forgetting previous data. Recent work by Ma and Perkins [25] on online novelty detection uses support vector regression but requires substantial amount of memory to store previous data points. Oh, Lee and Kote [29] discuss an algorithm for detecting motion on segments of frames and clustering these segments to identify normal events. Accordingly, anything not within these clusters forms an abnormal event. Their system lacks a comprehensive learning component for novelty detection, which is the focus of this paper.

Video Manga [41] is a new area of research that explores possible ways in which a video can be summarized and viewed as a comic book or Manga. Manga is a Japanese term that refers to comics. This paper outlines the need to derive effective clustering techniques and distance metrics to generate novelty mangas from large video repositories.

Searching images throughout the web normally produces many extraneous and miscellaneous images. Mukherjea, [27] explores the area of clustering where it can group the images together and represent images in a clearer fashion. This algorithm is based on low level color features such as the number of Red, Green, and Blue pixels in a region. They use the concept of major colors to derive a low-level description of the image. If the pixel count of a certain color is greater than a threshold, the image is then considered to have this major color. This is done for all the three major colors. Once this is completed for all the images, each image will then have a binary representation, where "101" means that they have the Major Colors of Red and Blue. These images are then grouped together with similar binary values. We use a modified extension of this clustering approach to group novel frames together. The grouping is required to determine the number of unique novelties that were observed within a video for a single scene.

2.2.1 Clustering Events vs. Clustering Novelty

Novelty clustering can be confused with event clustering. Within a video stream, anything that is found to be out of ordinary can be considered as an event. Typically, a particular motion sequence, or change in the background, or change in the still map of set of video frames is considered an event. For example, every car in motion in a video of an highway is an event. People entering or leaving an elevator generates an event. Turning a light bulb on or off creates an event in a still map. Thus a video scene is typically filled with lots of events. Clustering such events together based on image features is an easier task since complete information about the event is available either in the image feature space or the semantic space.

Novelty on the other hand is an event that is rare and unusual. It is difficult to determine novelty using a code-book of events since the variation between an event and novelty is not well-defined. Vehicles driving from left to right in a one-way street going right is not novel. A car going right-to-left in the opposite direction of this one-way street is 'novel'. The focus of our framework is to cluster novel events rather than any event within a video to determine how different machine based perception of novelty is from human perception of novelty.

Clustering based on novelty can also be confused with temporal event clustering. Even though the novel frames generated from the machine based system also include the frame number which denotes the time the novel frame occurred, it does not have any effect to the clustering procedure. We are attempting to cluster similar frames within a video together. Within temporal event clustering, the time each image is taken is used to determine the event it belongs to [10]. With this not being true for novelty in video, the novelty series can not be effectively represented as a time series. Lin and Keogh suggest that clustering of time series data is meaningless, but since novelty series is in itself a sampling of the raw video consisting of complete descriptions of what was found novel, clustering novelty does produce meaningful results unlike other time series clustering using a sliding window [23].

2.3 Sub-graph Isomorphism

This concept is fairly simple, suppose there are two graphs labeled as A(v1,e1) and B(v2,e2) where v1 and v2 represent the vertices. The edges are represented by e1 and e2. The problem is to find a sub-set of v1 and e1 which is in a subset of graph B. This problem has been shown to be NP-Complete [20]. A survey was done within this area to review other investigations [15][5][31][42]. Further review of each of these papers will need to be investigated later.

2.3.1 Generalized Matching Problem

Since this problem is NP-Complete, there has been numerous approximations done to reduce the complexity. Along with these approximations comes with restrictions and rules on they types of graphs which are used. Kirkpatrick et al explains that this problem is NP-Complete if the graph has at least three vertices [20]. They use the concept of scheduling to reduce conflicts within the second order. They investigate this problem using partitioning to split of the graph to quickly find the matching graphs. However, they have found to be NP-hard and found that their results were not as expected.

2.3.2 Planer Graphs

A planer graph is where the edges within a graph do not cross each other and only meet at the vertices. By having these types of graphs only, this paper claims that they can solve

the problem of sub-graph isomorphism in linear time. They also show that by using their algorithm, they can solve other types of graphs like disconnected graphs. Eppstein et al [15] claim that their algorithm may help other areas of sub-graph isomorphism based on some restrictions. One of the restrictions is that the tree width needs to be a certain size in order for it to be compared with to the pattern. Their results were positive and did show that it was performed at linear time.

2.3.3 Color Coding

This paper claims that by using color coding within the graphs, you could find simple paths along the graph, cycles, and small sub-graphs. Cycles are found by specifying the length of the cycle before hand. By finding a cycle in one graph, another similar cycle could be found in another graph within the same complexity. Alon et al [5] show that a bounded tree-width graph can solve the sub-graph isomorphic problem in polynomial time. They show that this technique is fairly simple to complete and can be applied to other areas of work.

2.3.4 SubGemini

Ohlrich et al [31] discusses the idea of circuits where the problem arises when you want to find smaller circuits among larger circuits. Circuits can be represented as graphs, thus being a sub-graph isomorphic problem. It uses Gemini's partitioning algorithm and claim that they can solve the problem very fast. This algorithm does not have any knowledge of these circuits, but do have information on how these are structured. They have shown that their results are faster than using brute force, but its intended for this specific application only.

2.3.5 Algorithm for Sub-graph Isomorphism

Ullman et al [42] was one of the first algorithms to find a better solution than by using the brute force method. It attempts to improve the algorithm by removing successor nodes within the graph. They mention that by implementing a hardware structure would improve these results, but haven't implemented it during their testing. The main improvement they have made within this algorithm is that it improves the time by using undirected graphs.

2.4 Time Series

Another form of Novelty Detection is called Time Series. Time Series is a set of numbers based on the order of time where the next number in the sequence is based on the previous values. This type of detection can find abnormal values that do not fit within the Time Series sequence. This area was surveyed to review different topics of Novelty Detection.

The algorithms and applications reviewed here are TARZAN [19], One Class SVM [25], Temporal Sequences [25], and Immunology [13].

2.4.1 TARZAN

By using the Time Series on the data, Keogh et al are able to find the "surprise" within the data [19]. From the values of the data, the paper attempts to find a pattern. Instead of finding a specific data or novel point similar to outlier detection, it finds surprising patterns. Older forms of Novelty Detection dealt with finding the novelty by Brute Force. This paper compares the pattern to a structure of the data by checking the frequency of this pattern over expected frequency. Since the time series is ordered, the pattern can then be converted into an alphabet. These strings of an alphabet form can then be represented by using a Markov model. This model is based on Markov chains where it is a sequence of events, where the probability of the next event is based on previous events. The algorithm TARZAN then uses Suffix Trees to search a set of strings. This allows the pattern to search only part of the tree that corresponds to the search string. If the number of characters that match is this tree is greater than a predefined threshold, then it is considered to be novel.

2.4.2 One Class SVM

An SVM can naturally detect outliers amongst different data sets shown by Ma et al [25]. This is a classifier that classifies data sets into their own separate objects. Initially it will create hyper-planes in a multidimensional space [1]. Then it will use Kernal functions to re-map the values or be considered as the Transformation phase. Then a simple linear function could then be used to separate two sets of objects.

Novel events are then found by using this model where the outliers are the novel events. It converts the time series into vectors where then it can be used for the SVM's.

2.4.3 Temporal Sequences

Novelty is based on a confidence level that is assigned to each novel event. It is similar to Time Series where it can detect any abnormal occurrences. It claims that Ma and Perkin's paper [25] simply just identifies the novel events without considering the whole set of data. Confidence is based on how much noise and error a novel event has.

2.4.4 Immunology

The Human immune system is a natural Novelty Detection system [13]. It automatically knows when a foreign cell has entered into the body. This novelty detection system is based on this concept. Just like the immune system, this paper uses time series to detect abnormal events.

This algorithm uses the category of "self". For a set of strings S, there are detectors that monitor "self" and anything that changes it would be considered abnormal or novel.

Chapter 3

Extracting Novelty

Monitoring surveillance videos manually to find an interesting event is often a tedious task. Surveillance administrators must review every second of the video since the timing of a novel event is unknown. These videos may contain hours or even days of uninteresting events before a novel event occurs. This approach may be easier by first detecting interesting events automatically and then presented meaningfully. Initially we review a algorithmic novelty detection system and then a human based system on perception. We explain how these twocan be compared together with various metrics. We attempt to improve these results by extracting regions to provide a better description of a single event shown in Phase 2 of Figure 1.1. These meaninful events are the clustered using our own algorithms. This is followed by our experiments and results for the comparisons between novelty detection systems using our clustering approach.

3.1 Novelty Detection

Surveillance administrators consider an event interesting if it is something new that they have not seen before. The administrator will learn about this event if it occurs repeatedly throughout the video and no longer consider this as interesting [35]. If this event does not occur as often, administrators will forget about this event and consider it interesting if it occurs again at a later time [21]. The VENUS novelty detection system uses this concept of learning which is based on two principles to detect novel events in a video stream.

The first principle detects an occurance of a stationary object if this object did not occur in the previous event. For example, a person leaving a black bag in the middle of an airport. If the occurance of the black bag has never occured, then this would be considered novel. This is determined by using a still saliency channel [3].

The still aspects found in a video are learned by averaging the low-level feature values on a topological map. This map represents the changes that occur based on feature values. The novelty is found by finding the average habituated feature value and comparing it to

the current feature value. If this difference is over a certain threshold, then it is found to be novel. In our example of the black bag, before the black bag arrived, the background was averaged to a certain value. Once the black bag arrived, the features gathered from the black bag caused the habituated average to change. This change caused the black bag to be novel.

The second principle is similar to the first, but it is able to detect novel moving objects. An example of motion novelty can be seen in Phase 1 of Figure 1.1. The first occurance of the white van moving from left to right is novel. Then the next occurance of the white van moving in the same direction has been habituated, thus not producing a novel frame.

This system detects novelty using a 8-by-8 pixel region within each frame of the video. Any visual occurance of this region is considered motion novelty. Within each region, clusters are generated based on the extracted features. If the new frame's region cluster can be merged to the current cluster, then a similar event has occured. A similar event must occur quite a few times for the system to learn and habituate on the event.

3.2 Comparing Human Perception with Machine-based Novelty Detection

It is common that we as humans shift our attention toward anything that is interesting to us. This process is a natural form of perception or novelty detection. Our goal in this experimentation is to formulize a strategy to capture human perception and compare it to the novel events by our machine novelty detection system.

We use an eye-tracker to detect how human subjects perceive novelty in video. The eye tracks were captured using an ASL head mounted ES501 system. Subjects were initially calibrated and then instructed to watch a series of videos. Though not restricted in any manner we requested that they keep their heads relatively still and focus on a 42 inch TV monitor. The eye-tracker tracks the subject's left eye to detect their eye movements. This system was chosen to best represent a person's attention because studies indicate that both eye movements and attention focusing uses the same parts of the brain [24].

There are five main steps for the implementing this human based novelty detection system. First the eye tracks are captured, then the scan-path corresponding to these eye tracks is extracted. The tracking data is segmented into pieces and the fixations are clustered. Finally the novel frames are isolated based on the fixation and eye-movement features.

Subjects were shown four videos each featuring a different environment. These videos were selected for their consistent viewing angle and relatively less motion. After the subject finished watching all four videos, their data was saved and converted to ASCII format for later processing. The actual process of this experiment setup is as follows:

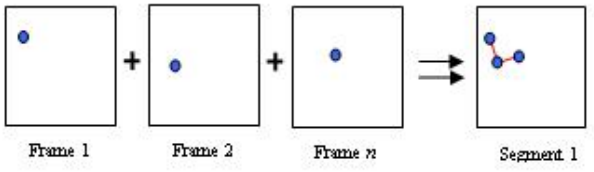


Figure 3.1: Blue Circle represents single fixation. Each frame has at least one fixation made by the subject. Based on the number of frames that is predefined beforehand, the frames are combined and clustered as Segment 1. (Best Viewed in Color)

- Step 1: The Eye Tracker was first calibrated to align the laser on the person's eye.
- Step 2: It took roughly five to ten seconds of test video to confirm the system was calibrated.
- Step 3: The actual test video was shown to the user.
- Step 4: The system then read the eye movement information as fixations within the video.

The subjects that volunteered for our study came from a wide range of backgrounds. Different backgrounds were needed to be ensure our results were not affected based solely on a subject's background.

3.2.1 Capture the Eye Track

Human eye tracks are recorded while the subject watches a video. The eye track data is then filtered and modified for calibration and readable representation. This will result in an easily parsable format for the attention finder algorithm. The format of this data file may vary depending on implementation, but this system requires at least X and Y coordinates across a time series. The scan path is extracted when the data file is read in and the linear scan-path that the users' eye followed during his/her session is cleaned and saved. This path structure is then analyzed in order to group fixations together into "Attention Areas".

3.2.2 Segment Eye-Tracking Data

Due to the temporal nature of eye track data and the fact that there will almost always be at least one fixation per frame of the video, we segment the data by a user specified amount. This will group together multiple fixations for a certain time period that we can group fixations that may or may not correspond with Attention Areas (Figure 3.1).

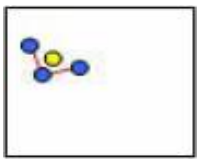


Figure 3.2: Yellow represents a centroid. Within a cluster made by the frames in Figure 3.1, a centroid is founded where this is the main fixation made for this set of frames.(Best Viewed in Color)

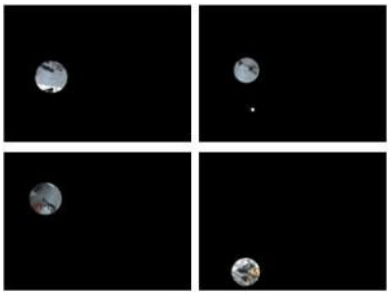


Figure 3.3: Fixation Example Output. Based on the centroids shown in Figure 3.2, the visible part is the region of attention. The black areas are the mask which is very similar to how the machine-based system shown its novel frames.

3.2.3 Cluster Fixations

The clustering technique used in this framework is simple but could be replaced by a more robust method. For each segment, the human based system determines the centroids of nearby fixations. This system uses a simple threshold measure to determine if fixations are "nearby" and thus belong to a particular cluster. Centroids (Figure 3.2) are created in order to determine the most likely center point of fixations that will later be used to create the mask that will highlight the Attention Areas and ultimately produce the novel image similar to the machine-based output.

3.2.4 Isolating Novel Frames

For each centroid in a segment, the Attention Area is determined and a binary mask is created. This binary mask is applied to the corresponding frame from the video and is saved. The effect of combining the binary mask with a frame is to "black out" all areas that are not found as being novel Figure 3.3. The blacked-out areas correspond to the mask while the visible regions are the Attention Areas for each of the segments processed. The human-based system uses a simple rule, (size of area corresponds directly to number of fixations used to determine the centroid) for determining the actual Attention Area but could be extended to use a more sophisticated measure. This system currently uses the first

frame of the current segment to apply the mask to. At this point we are unsure if this is a good way to do it.

We have applied the connected component region and clustering algorithm to the human detected novel set. Our experiments show the region based clustered index performs better as in our previous results.

3.3 Designing Distance Metric for Human and Machine Comparisons

Once novel images are found by both human and machine systems, a metric was needed to determine how similar they were. Numerous forms of distance metrics are available to determine the distance between two vectors. Where each vector is simply a set of features for each novel image. Instead of simply using these features alone, this system uses two new forms of metrics called location and feature similarity.

3.3.1 Location Similarity

The first metric used by this framework is location similarity. Novel frames usually contain only a small amount of novel area. The task here is to extract the actual location on the image where novelty is taking place and to compare that with the location of novelty from a corresponding frame in order to determine whether or not the novelty detected by both systems captures the same locations. Novel location regions are extracted from the images using a recursive dissection technique. The algorithm breaks the image up into sub-regions until a specified depth using quad-trees [8]. This not only quickly locates novel regions, but also creates a hierarchy of such regions. This hierarchy can then be used later for further feature comparison, as we will see for the feature similarity metric. Figure 9 shows the comparison of similar regions between two novel images and they're scores.

3.3.2 Feature Similarity

The second metric is feature similarity. This is an abstract concept and can therefore be used for any type of comparable feature. In its implementation, it looks at pixel color as the comparable feature. Using the mean hue scores generated by the feature extractor, regions of an image are compared to regions of other images. The regions to be compared have been previously generated via the location similarity metric detailed above. Using the hierarchy of similar regions, the feature similarity metric is able to identify regions across images that are similar to each other. The user can specify how detailed of a comparison to perform, which is useful for fast comparison across very different images as well as detailed

comparison between images that are very similar. Figure 3.4 shows scores for the feature similarity metric. Figure 3.4 shows a visualization of the distance metrics. The second image is compared to the first, while the third image shows the novel regions as well as the intersection of those regions. Scores are calculated based off of the two metrics and printed on the middle image. As you can see, although the first image has a 0% location similarity to the second, their features are still 98% similar. Likewise, the features of the second comparison are 99% similar, with a location similarity of 15%.

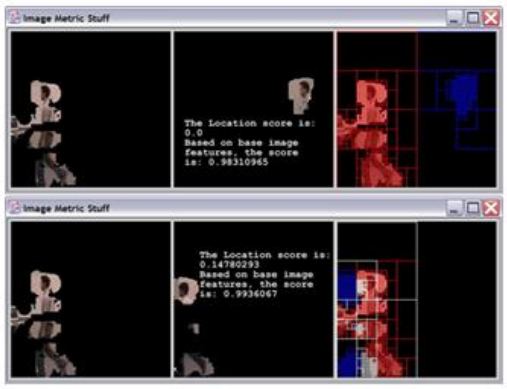


Figure 3.4: Novelty Location and Novelty Features based similarity metrics. Left-to-Right: The first frame image is machine-novelty. The second frame image is human novelty. The third is their difference. The top image shows that the novelty location distance between human (blue) and machine (red) is large since the two novelties were not detected within the same region. The lower image suggests that the distance is small since the two novel images overlap each other. Their scores in color feature space are similar.

3.4 Novel Region Extraction

Connected component regions are areas within the image that are related. Once our novelty detection system has detected a novel key frame or non-region, the image may contain more than one connected component region. Since its hard to quanity a single event over a novel key-frame. Our goal is to extract these components to identify a clearer event. Phase 2 of Figure 1.1 shows a novel key-frame describing a white van and people walking. These are two seperate events and regions of each must be extracted to distinguish the two. In this paper, the results section shows that regions provide better performance in terms of quality over non-regions.

Bounding box regions are found by initially converting the novel key-frame into a binary format based on a threshold. The threshold will distinguish the black areas found in Figure 1.1 as the value of 0 and the other areas as 1. The areas with the value of 1 are bounded by a square region. We then label each square region to determine the number of components within a key-frame. For each component, we can find the top leftmost (x,y) coordinates

along with its width and height dimensions (Figure 3.5).

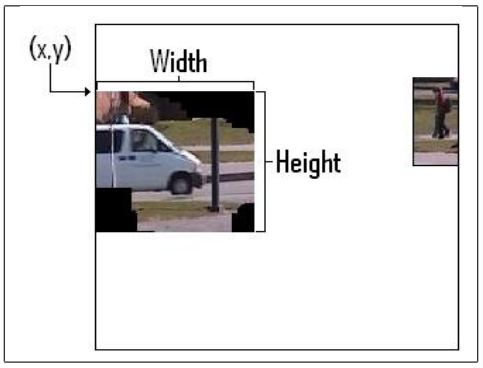


Figure 3.5: Region component coordinates for (x,y), width and height.

Once the regions are found, the region needs to be extracted from the key-frame for both visual and analyzing purposes. Traditional image formats such as JPEG, GIF, Bitmap, and Tiff are unable to allow extraction of a region from a compressed image. A new and emerging technology that allows this type of extraction is called JPEG 2000. Based on the coordinates found, we can extract two seperate components from the image in Figure 3.5.

3.5 Clustering Novel Regions

By its very nature, video normally contains a large amount of novel events. Some will contain more than others. For example, surveillance video of a basement warehouse will most likely contain less novel events than say an action movie. When many novel frames are extracted from a video, it is helpful to summarize novelty frames with which to browse the novelty set. For this reason, this system uses a modified algorithm described by Mukherjea [27] to summarize the data.

3.5.1 Feature Extraction

Each region describs a certain event occuring within the video stream. The description of each event was found automatically using low level features. The feature set we used are Hue, Saturation, Value (HSV), orientation, and intensity. HSV is selected instead of Red,Green,Blue (RGB) due to its similarity with the way in which humans perceive color [50]. Orientation is described as the number of pixels found at the 0, 45, 90, and 135 degree plane. Intensity is based on the number of pixels found over a certain threshold. Our results shows the features are better focused by than by a single key-frame.

3.5.2 Clustering

As an event 'habituates', multiple frames of the same novel event may occur where each subsequent frame will lose the novelty of the event. It is necessary to group these frames

together as a single event to simplify the novel data set.

Initially the features are extracted from the novel key-frames or connected components. Based on [27] algorithm, the average of each feature is set as its threshold. Since novel frames can exist above and below the average with a short distance and be placed in separate clusters using this algorithm. In our algorithm, an upper bound of the confidence interval was used as the threshold. This is ensure that novel frames near the average are placed in the same cluster. Either the lower or upper bound could have been used as part of the threshold. Once the threshold is established, each feature of the novel image was compared against this threshold to discretize the feature vector. If the the feature value is greater than the threshold, its assigned the binary value of 1 otherwise a zero. These binary values are then clustered together. This algorithm is summarized as follows:

Algorithm:-Modified Feature Based K-Means
INPUT: Novel Frames
OUTPUT: K Clusters

Step 0: Setup and Initialize.

$P_{ij} = number\ of\ pixels\ of\ jth\ feature\ in\ ith\ image$

$where\ i = 1, 2, 3, ..., n\ and$

$j = 1, 2, 3, 4, 5, 6, 7, 8\ (1 = Hue, 2 = Saturature, 3 = Value; 4-7 = Orientation; 8 = Intensity)$

Step 1:Summation of feature values over all images

$\S_j = \sum_{i=1}^{n} P_{ij},\ where\ j = 1, 2, 3, 4, 5, 6, 7, 8$

Step 2: (Determine threshold t_j for $j = 8$ feature)

$\bar{x_j} = (\frac{\S_j}{n}),\ where\ \bar{x_j}\ is\ the\ mean\ value\ for\ feature\ j$

$Z = 1.95\ (95\%\ Confidence)$

$\sigma_j = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x_j})^2}$

$t_j = \bar{x_j} + Z(\frac{\sigma_j}{\sqrt{n}}),\ where\ j = 1, 2, 3$

Step 3: (Discretize each image vector)

$b_{ij} = 1\ if\ P_{ij} > t_j$

$= 0\ if\ P_{xj} \leq t_j$

Step 4: (Cluster each image based on Binary value)

$C_k = k\ for\ i = 1, 2, 3, \ldots, n\ and\ k = 1, 2, 3, \ldots, 8$

$$
\begin{aligned}
where\ k\ &=\ 1\ if\ (b_{i1}=0, b_{i2}=0, b_{i3}=0)\\
&=\ 2\ if\ (b_{i1}=0, b_{i2}=0, b_{i3}=1)\\
&\quad\ \vdots\\
&=\ 8\ if\ (b_{i1}=1, b_{i2}=1, b_{i3}=1)
\end{aligned}
$$

3.5.3 Sequence Clustering Algorithm

An interesting method of creating clusters is to identify the start and end point of each of the novelty that occurs within a video. Instead of comparing all the novel frames within a video to find similar frames. The sequence of the novel frames that are similar is clustered. For example, if there were 5 novel frames found within a video each occurring right after the other. If the distance between frame 1 and frame 2 are under a certain threshold, then put within the same cluster. If frame 3 and 2 are greater than the threshold, then generate a new cluster. This will create sections within the video stream of when a new novel event occurs. For its simplicity, we used the Euclidean distance to compute the feature-based similarity between subsequent frames. The Sequence Clustering Algorithm (SCA) is summarized below:

Algorithm:-Modified Sequence Based Clustering (SCA)

INPUT: Novel Frames

OUTPUT: k clusters of sequential frames

Step 0: (Initialization)

Sort the images in ascending order of sequence

based on frame number where,

$$f_i\ \in\ f_1, f_2, \ldots, f_n$$

$$\forall i\ <\ j,\ f_i\ <\ f_j$$

Step 1: (Euclidean Distance in terms of features for subsequent frames)

$$d(i, i+1) = \sqrt{(P_{i1} - P_{(i+1)1})^2 + \ldots + (P_{i3} - P_{(i+1)3})^2}$$

while $i = 1, 2, 3, ..., n-1$

Step 2: (Determine Threshold t)

$$\bar{d}\ =\ \frac{\sum_{i=1}^{n-1} d(i,i+1)}{n-1},\ where\ \bar{d}\ is\ the\ average\ distance$$

$$\sigma\ =\ \sqrt{\frac{1}{n}\sum_{i=1}^{n}(d(i,i+1) - \bar{(}d))^2}$$

$$t\ =\ \bar{d}\ +\ Z(\tfrac{\sigma}{\sqrt{n}})$$

Step 3: (Clustering)

Let $i = 1,\ k = 1. f_1 \in C_1.$

$$foreach i:\ if\ d(i, i+1)\ <\ t\ for\ i = 1, 2, \ldots, n-1$$

then $f_{i+1} \in\ C_k$

else $k = k+1$ (*new cluster*) *and* $f_{i+1} \in\ C_k$ k = number of possible clusters. If each image forms its own cluster then k=n.

3.5.4 Representative Image Algorithm

Although our clustering algorithm provide a clustering, they do not present a good way of representing each cluster. This algorithm uses the pixel scores over the entire cluster to first get the mean values for each color. One can think of this mean vector representing the hypothetical centroid for that cluster. Next, for each image we compute its distance from this centroid. The image whose features have the smallest distance from the centroid is used as the representative image. This is described below and the notation is the same as our clustering algorithm.

Algorithm:-Determine Representative Images
INPUT: Novelty Clusters
OUTPUT: Representative Images

Step 0: (Initialize and Pre-process)
$C_k = k^{th}\ cluster\ consisting\ of\ x_1, \ldots, x_{g_k}\ frames$
$where,\ g_k\ is\ number\ of\ images\ of\ the\ k^{th}\ cluster.$

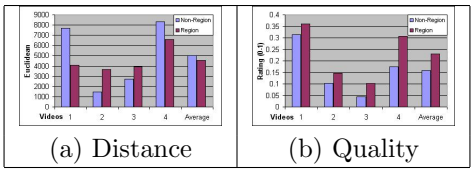
Step 1: (Compute the Mean)
$for\ each\ cluster\ 1, \ldots, k:$
$for\ each\ feature\ 1, 2, 3, 4, 5, 6, 7, 8:$
$compute\ \bar{x_{jk}} = \frac{\sum_{i=1}^{g_k} x_i}{n},\ where$
$g_k\ is\ number\ of\ images\ in\ the\ k^{th}\ cluster.$

Step 2: (Compute Euclidean distance)
$d(x_i, \bar{x_k}) = \sqrt{(x_{i1} - \bar{x}_{1k})^2 + (x_{i2} - \bar{x}_{2k})^2 + (x_{i3} - \bar{x}_{3k})^2}$
$for\ images\ i = 1, 2, 3, ...n\ and\ k\ clusters.$

Step 3: Determine centroids as the representative frame
$r_k = min(d(1, \bar{x_k}), d(2, \bar{x_k}), \ldots, d(n, \bar{x_k})$
$for\ each\ cluster\ k = 1, 2, 3...$

3.6 Experiments and Results

The most apparent difference that was observed between the human and machine novelty was in the rate of habituation for novelty detection. Since, the machine-based system uses a habituation technique where new novel events are "remembered" for a while, the significance of novel events wrt the current novelty within the video degrades over time. As an example



(a) Distance (b) Quality

Figure 3.6: Machine Novelty Detection vs. Human Perception: Region and Non-Region comparisons

consider a video of an airport ticketing area. Initially, the people walking around the airport would be very novel and the machine-based system will display them as novelty within the scene. Over time, the machine-based system will learn that people walking around is not novel. However, in a situation where some person drops a bag on a chair and walks away the still map will detect the bag as being novel. A lab environment video for this and several such machine novelty detection scenarios are available for review from our website as mentioned before. Consequently, as our video mangas for machine-novelty indicate, the system is rather exhaustive and generates several novelty clusters once the novelty metric as been pre-defined with an appropriate habituation function. Marsland et al observed that humans too tend to have this type of habituation [33]. Our study validates this fact. We observe that human subjects tend to pick up new novelty within the video very quickly and, if it is not a major event, return their focus of attention to the previous attention area just as quickly.

3.6.1 Machine vs. Human Perception

Our machine based novel data was compared to human perception to measure how close machines are to humans in terms of novelty. We experimented with 10 subjects that viewed four surveillance videos and compared it to the results generated by our machine novelty detection system. Both machine and human results were clustered and the best representative images were found. Using the machine's results as the basis, we found the shortest Euclidean distance between each machine's to the human's representative images. Quality was also measured to determine if the machine's and human's pair of images were similar or not.

Chapter 4

Indexing Novelty

Before we can begin analyzing state-of-the-art indexing algorithms, fundamental indexing schemes were needed to be studied to provide a basic background to indexing. Initially we developed a database containing novel frames using three fundamental indexing schemes which were the Hash table, B+tree, and the R-tree. This system's interface was a SQL parser that could query, insert, delete, and update using these schemes on a spanning block file system. Spanning blocks allow a single record within a table to be broken up into two or more blocks to allow the whole block to be filled. We analyzed the cost estimates for each type of command for all three indexes. This provided us sufficient background to continue with an analysis of four different multimedia indices and our own approximate clustered index. We provide results at the end of the chapter to compare and contrast the utility of clustering, regions, and the use of the different multimedia indices.

4.1 Indexing Novel Frames using Spanning Blocks

4.1.1 Indexing Schemes

The three indexing schemes we used were Hash Table, B+ Tree and R Tree. Each scheme implemented had its own issues that we had to deal with. This section discuss our implementation of the three indexing schemes and also briefly discusses the SQL Parser that was used in this implementation.

Hash Table

In hash based index the data entries are kept in buckets. And each bucket is a collection of one primary page and zero or more overflow pages. When the search key value is given it is easy to find the bucket using a hash function which is denoted by h and the value of h(K) is the address of the desired bucket. The Bucket size could be the number of pages that can be held at the same address. There are two types of hashing.

Static Hashing: The numbers of pages are considered to be fixed and the pages are

allocated sequentially. Performance degradation can occur due to long overflow chains and it can be avoided by using Dynamic Hashing Technique. The two kinds of dynamic hashing techniques are Extensible Hashing and Linear Hashing. In extensible hashing reading and writing are considered to be expensive.

In the project, hash based indexing is implemented were the key was the column value and the bucket contains the line numbers which points to the flat file. The algorithm used to implement different DML statements have been discussed in following sections.

Load

When the user runs the application it first loads the indexes into the Hash Map data structure. An array of objects for each attributes were created and loaded into the memory at run-time. Each attribute hash map object contains key as the value of the attributes and value as the line number. When there is more then one tuple with the same value the line number gets added. To do that a vector is taken and for each common value the line number is added into the vector. The main reason for storing line number is to improve the performance and access rate. So at run time in loading phase all the tuple line numbers are stored with appropriate key. When the user poses a query based on the attribute value line number is retrieved.

Insert

When the user poses a query to insert the tuple into a flat file it also has to update the Index. To update the index, the value of each attribute is taken and if the attribute value is already present as a key then its value, which is a vector, is updated with the current line number and if the key is not present then a new key is added with a current line number. The tuple is also written on the flat file.

Query

When the user poses a select query, based on the given condition the tuples rather line numbers are retrieved from hash map. If the selection is a single condition then all the line numbers are obtained and then for each line number the specific tuple is retrieved from the file. The line numbers are the pointer to the record in the flat file. If the query contains multiple conditions then initially all the line numbers for the first conditions are retrieved then all the line numbers for the second condition are retrieved and stored in two different sets. Then intersection of two sets will return us the line numbers, which obeys, AND operator condition. Similarly for the OR condition union of two sets will return the line numbers and then we can access the required attributes, which needs to be displayed.

Delete

When the user poses a query to delete a tuple from the flat file, the Index also needs to get updated. To delete from the Index we need to remove the line number from the vector. This should be done for all the indexes assigned for the given database. To delete from the file we need to get the line number and then a tombstone ['D'] is kept in place of that record.

Update

When the user poses update query, the Index for the given attribute gets updated. Initially the line number is obtained from the query and then update of the index takes place. Index is updated by creating a new key or if it the key is already present then the vector of line numbers is updated. Based on the performance measurement we could see that hash map is best for equality selection and not for supporting range searches.

B+ Tree

B+ Trees are similar to B Trees with each node containing a set of keys and pointers. The height of a B+ tree grows or contracts dynamically in response to insertions or deletions. The B+ tree however always remains balanced (i.e. the number of levels from root to leaf for each leaf always remains the same).

The B+ tree also had a property called the fill factor (i.e. 50%) which defines the percentage of a node (except for the root node) that must be occupied at all times. If the occupancy of a node falls below 50% or the minimum occupancy, the node is made to combine with another node. If of the other hand, the occupancy of a node increases beyond the maximum occupancy, the node has to split into two nodes. The sections below describe how we implemented the B+ tree indexing scheme in our case.

Implementation

We made use of the Generalized Search Tree (GiST) package developed by University of California, Berkley and the B+ Tree (BT package) implementation of it which was developed at the computer science department of University of Cape Town, South Africa. Both packages were written in Java and were available off the internet. We were therefore to able extend their existing functionality to suit our case.

The GiST package is a generic package which can be used to implement any type of tree structure. This include Binary Search Trees, B+ Trees, R Trees, hB Trees, TV Trees, Ch Trees, Partial Sum Trees, etc. The BT package inherits from the GiST package and extends its functionality and implements the structure of a B+ Tree. We extended our B+ Tree class from the BT package to represent the B+ tree we used in our implementation. The B+ Tree used in our implementation had a minimum occupancy of 2 and a maximum

occupancy of 4.

The B+ Tree indexing scheme was implemented for both direct indexing of the flat file as well as for indexing using spanning blocks. For the direct indexing of flat files, the implementation makes uses Random Access to the flat file. With Random Access, individual tuples can be extracted from the file using a pointer to the tuple without reading the entire file there by saving on I/O reads and writes to files. When using blocks, BufferedReader was used to read the entire block, as blocks normally represent the smallest units and the whole blocks should be read a time.

The B+ Tree implementation also calculates the performance statistic for each type of query during the execution of the query. These statistic include time taken well as total reads and writes in term of tuples, blocks and bytes. Finally, the B+ Tree index implementation makes use of tombstone character 'D' to specify a deleted line in the flat file. Details of some of the other features of the B+ Tree indexing scheme are highlighted in the sections below.

Load

While loading the indexes in the B+ Tree implementation, separate B+ Trees are created and initialized to index each one of the attributes of the Still and Motion tables. The B+ Trees are initialized to having a minimum occupancy of 2 and a maximum occupancy of 4.

The attributes are indexed by creating a pointer to each tuple in the flat file or blocks. When indexing the flat files directly, the key is the value of the required attribute of a record, while the pointer contains the line number of the tuple in the flat file where the record value exists. When indexing using blocks, the pointers contains the ids of all the blocks over which the tuple containing the required record value spans

The B+ Trees for each attribute are stored in an array. The index in the array where the B+ Tree of a particular attribute is stored is the same as the index value assigned to the attributes of the tables. Hence the index on an attribute can simply be called by referring to its index value in the array.

A typical B+ Tree index therefore consists of a root pointing to sub nodes, and the sub nodes pointing to the leaves of the tree which store data in terms of pointer the tuples stored in the flat file or blocks. While loading, any tuple specified with tombstone character 'D' (i.e. marked deleted) is simply skipped.

Insert

The B+ Tree index supports insertion into both Still and Motion tables. Insertion only works with direct indexing of the flat files and not with blocks. The insert creates the required new tuple and using a pointer to the end of file in the flat file, inserts the tuple at the

end of the flat file. This is done using Random Access to the file. It then iterates through each attributes B+ Tree index and updates it with the new tuple.

Select Query

B+ Tree indexing scheme supports select queries for both Still and Motion tables. It also support complicated select queries with an AND or OR in the Where clause of the query. The query works with both direct indexing of the flat files as well as when using indexing with spanning blocks.

While executing a query, the Where clause condition is first evaluated and pointers to all the tuples that match the condition are found and stored in a collection. If an AND condition is specified in the Where clause then the tuples that match the condition for each attribute are first found and stored in separate sets. An intersection of the two sets yields the collection of required tuple pointers. In the case of an OR condition in the Where clause, a union of the sets is found instead. All the attributes specified in the select clause are then displayed from the tuples.

Delete

B+ Tree indexing scheme supports deletes for both Still and Motion tables. It also supports complicated queries with an AND or OR in the Where clause of the delete statement. The query works with direct indexing of the flat files only. While executing a delete, the Where clause condition is first evaluated and pointers to all the tuples that match the condition are found and stored in a collection. If an AND condition is specified in the Where clause, then the tuples that match the condition for each attribute are first found and stored in separate sets. An intersection of the two sets then yields the collection of required tuple pointers. In the case of an OR condition in the Where clause, a union of the sets is found instead. All tuples that are pointed to are then deleted from the flat file and replaced with a tomb stone character 'D' at the beginning of the tuple. The occurrence of that tuple in indexes of all the attributes is also deleted.

Update

B+ Tree indexing scheme supports updates for both Still and Motion tables. The update works with direct indexing of the flat files only.

While executing an update, the Where clause condition is first evaluated and pointers to all the tuples that match the condition are found. The pointers are then used to access the various tuples in the flat file and overwrite them with new values.. The tuples are also updated in the B+ Tree indexes for all the attributes.

B+ Tree Printer

As an additional functionality, we also implemented a B+ Tree Index Printer class. The class build a B+ Tree Index on a specified attribute of Still or Motion table and then display it on the screen. Using the B+ Printer, one can view the internal structure of the B+ Tree for any of the attributes of Still or Motion table. This includes the number, level, item stored or pointed to by the root node, the sub nodes and the leaves.

R Tree

R Trees are very similar to how a B+ Tree is laid out. They both have very similar structures. The advantage of a R Tree over a B+ Tree is that it can support range queries. In R Trees, it can store data within the index in the form of a coordinate. There can be two attributes where one attribute can be specified as the X coordinate and the other attribute as the Y coordinate. Likewise, a single attribute can represent both the X and Y coordinates. By allowing the use of two attributes as two coordinates, it would achieve joins fairly quickly. Especially in multimedia tables in our project, this index may be useful. But there were numerous issues within using this type of an index in our project discussed later in this report.

Load

There were two options we analyzed for doing the load of the R Tree. The first was the same as the other two indexes by indexing it on each of the attributes for both the motion and still table. The other option was using R Tree as an advantage and index it on two attributes, which will make the search time much faster. The issue with this is that in an R Tree, it is expected that you would know which pair of attributes would most likely be used and index on only those pairings. In our case, any combination of attributes or single attributes should be handled. If we wanted to do a pair of attributes, since there were 11 for still and 6 for motion, there ended up to be close to 140 different indexes to be stored within memory. This would be a major scalability issue if we decided to add more attributes in the future. Thus we decided to use an index for each attribute alone.

Insert

The insert of an R Tree is similar to the B+ Tree, but they can be inserted in the form of a coordinate. Since we're indexing each single attribute and not indexing on a pair of attributes. Each attribute will be both the X and Y coordinates of the insert. Along with each insert, the line number of where this records is stored within each key values of the index. For any conflicts that occur within this index, a vector is used to store the multiple line numbers in a string format.

Query

Querying using an R tree shows the best performance out of all the other two indexes. Further results are shown in the Results section. Since the attributes are in the form of coordinates, we can define a minimum (X,Y) and a maximum (X,Y) coordinate to find a range within an attribute. These min and max coordinates find a sub-space in the form of a square which extracts the information fairly easily.

Joins

Joins were performed by using a "trick" that could be used based on the data that was given to us. Since the still and motion data are extracted from novel frames within a video, this data will always be presented in sequential order. Within the still table, the primary key is the ID and the index value store for each attribute was the line number. Since the line number is essentially like the ID, for each attribute within the index, the line number for that attribute can also represent the ID the attribute is associated to. For the motion table, its very similar to the still table. Since everything is in sequential order, the line number can also represent the started attribute. Since the endid attribute is always seven plus the started, we can calculate the endid without looking within the file. Since we can find the id, started, and endid from looking at the index itself, we can then determine if the still id is within the range of the motion's started and endid. Thus joins were accomplished. In our development, this was simply proof of concept and R Tree was the only index where joins were developed.

Delete

Similar to the other indexes, delete is simply deleting a key value within the node. For our implementation we used the GIST package along with help from Yan Hu to develop the R Tree. There was a bug within the code which when a delete occurred, it would delete all occurrences of the key value along with the duplicates. The code was modified to also send the line number the key was associated to so it would delete only a single occurrence at a time. Similar to other indexes, a tombstone was added to the file when a delete occurred.

Update

There wasn't a direct method of updating the R Tree by using the GIST packages that was provided to us. Instead a query was made through the RTree to find the old key values. Then these values were deleted and the new updated tuple was inserted with the new line number. This is very similar to how the other indexes updated an index.

SQL Parser

The basis of our SQL parser was the use of the ZQL Parser [1]. This ZQL parser took care of parsing the important information from each of the SQL commands. There were extensive error checking done within the Select, Insert, Update, and Delete commands. This parser was used for all the three indexes which provided error checking to ensure the data received by the user was syntactically correct.

4.1.2 Spanning Blocks

This project we went through three phases before we implemented Spanning Blocks. The first phase was a simple cost analysis program where we outputted the number of blocks we read and wrote within the program. The second phase dealt with mimicking the file I/O system by transferring the data into blocks specified by the user without spanning blocks. The third phase was to implement and allow for spanning blocks.

A Simple Cost Analysis

Our first take was to determine the number of blocks we were reading and writing during numerous SQL commands. We used our textbook as the basis of this analysis. Within the textbook it defined each tuple or record to be the size of one block. We could see by using this technique which commands read and wrote the most. The results gathered from the SQL commands were as expected, the output of records was the same as the number of records read and written. The load block number was expected since it needed to read all of the blocks and load the index into memory beforehand.

Blocks without Spanning

Our second task was to transform the single flat files of motion and still into a user defined block size. For example, the user could specify a 512 byte block. Since each record in still is a fixed length of 132 bytes, 3 records can fit into each block. Also, in order for a block to be 512 bytes, we included a block header of 10 bytes. This header was used for the file name, since the files were called "10.txt", it accounted for 5 bytes of the block. If the user chose 512 byte blocks, we would allot 502 bytes and 10 bytes as header. We had to also change our method on how we stored the line numbers within the index. Since a records may belong to any block we had to remember the block number and the where the records was within the block. We stored a string value of "¡block¿¡space¿¡line¿". The block number was the block file the records was stored in. The line number represented the location of the record within the block.

When we retrieved the information from the block, we attempted to mimic the file I/O system. For each block, we read all of the records into memory. Then we would find the

record by using the line number stored in the index. It would be easier to directly access a line number since we already know it, but this would be cheating and we are not allowed to read a block partially.

The method of using blocks were meant only for querying. The other SQL commands were not integrated with this design. But it would be quite simple to integrate it. It would have to incorporate the block scheme and all the information needed are located within the index.

Spanning Blocks

The final task of the project was to incorporate the concept of spanning blocks into our design. Spanning Blocks occurs where there isn't enough room for a full record to be stored within a block and it needs to be split up onto a set of blocks.

There were a couple of rules that had to be defined before we could implement this concept. Each attribute within both tables could not be split up if there was room within the block. This means that if the value for red was "123456" and there were only 3 bytes left. We could not put in one block as "123" and in the other block as "456". If there wasn't room for a full attribute, it needed to go on the next block. Another issue was that the user could not specify a block size less than 12 bytes. Since an attribute cannot be broken up, there will not be any data stored within blocks less than 12 bytes.

We also had to change our design on how we stored the string value into the index. Since there can potentially n number of blocks for a certain record. We used this regular expression to use to store in the index: "(¡block¿¡space¿¡line¿)+ ¡space¿¡id¿)". Block and line are used the same way like our previous design. ID is used strictly for joins only which will be described later. An example of using this scheme is let's say there's a record that fits into three blocks. In each of these blocks the part of the records sits on the first line. Also this records represents 5th id. Then the index would have a string of "1 1 2 1 3 1 5". This says that it sits on the first block, first line number, then the second and the third. Each record stores this string within the index. Once a query occurs, a string like the one above will be the index value. We would simply go through the string and find the first block and read the records, and then combine it with all the other blocks, then having the full tuple to use.

A counter was used to store the id number at the end of the string. Since we cannot use the line number to represent the id for the joins. We added a counter value to mimic the line number or the id. By using this we were able to perform joins using spanning blocks. Again, queries were only used for this concept. By reason of proof by concept, we implemented this using B+ and R trees. For Joins, we only used R Trees since B+ Tree was not used as part of the Join design.

4.1.3 Cost Estimates

When a query is posed, the parser parses it and it gets transformed into logical query plan. Logical query plan is then converted into physical query plan. Different physical plans are derived from the logical query plan. Each physical plan estimates the cost of executing it. The one with the least cost is selected and send as an input to the query execution engine.

Cost estimation for the plan can be based on parameters of the data that can be computed exactly from the data or by a process of statistics. Reasonable estimates of relation sizes can be used to estimate the cost of the physical plan. The conventions we used in estimating the cost are B(R) which is the number of blocks needed to hold all the tuples of Relation R, T(R) is the no. of tuples and V(R, a) is the value count for attribute a of relation R.

Projection

Projection is considered to be different from other operators, in that the size of the result can be computable. For estimating the size we take the size of each attribute and sum it along with header size of each tuple. We can calculate the number of blocks required based on the size of each tuple, number of tuples, and the block header information. The projection shrinks/grows the relation by change in number of arguments.

Selection

In selection we reduce the number of tuples required though the size of each tuples remains same. In simplest kind of selection where attribute is equated to a constant then we can estimate by the number of different values the attribute as. It can be computed as the number of tuples in the relation by no. of selected tuples which contains the constant.

Estimation can be done by this formula T(S) = T(R)/V(R,A) where T(R) is the number of tuples and V(R,a) is the value count.

Estimation for inequality is done by taking one third of the relations' tuple as it assumed that inequality tend to retrieve a small fraction of the possible tuples. The formula for the inequality is T(S) = T(R)/3 When the selection is not equal we consider all the tuple except one. The estimate can be taken as either T(s) = T(R) or T (S) = T(R)(V(R, a)-1/V(R, a) which is slightly less as an estimate.

When the selection condition is AND we can do the estimation by breaking the condition into selection with one condition and then multiply it by the selectivity factor for each condition.

When a selection involves an OR condition the size of the result is the sum of the tuples which satisfy both of each. The other way of doing the estimation is calculate the fraction of tuples, which do not satisfy condition 1, and then the fraction of tuple, which do not satisfy

the condition 2 and the product, gives the no. of tuples that are not in S. If we subtract by 1 we get no. of tuples which are in S.

For union the size can be the sum of the size of the arguments or average of the sum and the larger. For intersection it is no greater then the smaller of the two relations. When the selection is difference the result can have between T(R) and T(R) - T(S) tuples.

Joins

Another form of cost estimation is using Natural Joins. A natural join is where two relations are joined by a common attribute. Joins are estimated by a simple concept. Let's say we have two relations R and S. Where each has a common attribute of Y. The formula is T(R)T(S)/max(V(R,Y),V(S,Y)). Where T(R) is the number of tuples in relation R and T(S) is the number of tuples in S. V(R,Y) is the distinct values of R for attribute Y. It's the same for V(S,Y) but for distinct values of R. This can be done for more relations than just two tables. Simply join any of the two, then start joining the next relation.

4.2 Indexing of Novelty

In order to effectively represent novelty data, we have surveyed numerous R-tree varied implementations to determine the best index for our query engine. The R-tree was originally designed by Guttman [17] to access spacial data. Since our framework uses regions found in novel images, the regions and feature information can be represented spatially which directly relates to the R-tree. This tree is an extention from a B+trees in multidimensional space and a height balanced multi-way tree. All the leaf nodes in this tree are at the same level and can perform point and range queries. The performance of a R-Tree is based on the coverage and overlap of rectangular regions. The minimun rectagular bounding is the coverage set of nodes. Overlap is based on the intersection of these rectangles.

In our original indexing scheme, a R-tree was used to index novel data from a video stream. Each image has eight different features that are needed to index. By using a R-tree, its often hard to represent an eight dimensional R-tree visually. Thus, a single R-tree was used to represent each attribute, where eight R-trees became an eight dimensional R-tree indexing scheme. A query was performed using this index by searching all eight R-trees individually to retrive the most similar image to the query image.

However, there are some disadvantages using this scheme with clustering. The clustering algorithm explained in the previous section was performed on our novel data set. For each cluster in the data set, there is a single index based on eight R-trees. This created eight times the number of clusters of R-trees for which the system had to maintain at any given time. In order to reduce the amount of indicies in a system, the number of R-tree, and utilize the clustered data, we used our own Approximate Clustering Indexing Algorithm.

4.2.1 Approximate Clustered Indexing

The goal of this indexing scheme is to improve query time and have better retrieval quality than our original design. In our original design, we implemented an eight dimensional R-tree indexing scheme. Instead of using eight R-trees for a single index and since our data is clustered, only a single R-tree based on one attribute is needed for an index. We already know that the images in a cluster is similar, so only a single index is needed. In order to compare this scheme effectively, we used our original design but with approximate indexing on non-clustered data as a benchmark. Our results show that the Approximate Clustered Indexing algorithm performs better in terms of speed and quality.

We used this framework on top of a R-tree algorithm. We also uses this framework on a number of state-of-the-art indexing algorithms except for an applied Singular Value Decomposition based algorithm which is explained in the next sections. Our of the many we surveyed, we chose four that is the most interesting and benificial to our query engine.

The first algorithm we surveyed is called the Pseudo PR-tree designed by Arge et al [7]. This indexing system utilized the extreme points in a data set. In order to obtain the minimum and maximum values of an R-tree, a rectangle bounding box covering the whole tree is needed which can be very expensive. Arge et al [7] have modified the R-tree design to handle this types of data. The second indexing algorithm we looked at used Singular Value Decomposition on a mult-variable data set. It is intended to decompose a large $m\ x\ n$ multivariate data matrix where $m \geq n$ to a smaller data set set without reducing the variability between data points. These points are stored in a suffix-like tree which can then incorporate all eight features in a single tree. This is the only indexing scheme we reviewed that did not need to use our Approximate Clustering Indexing algorithm framework.

Our third index is based on viewing 3D objects at a perspective efficiently using a HDoV-tree designed by Shou et al [34]. Even though our data is set is not of the 3D form, we have applied the ideas from this scheme and retrieved interesting results. Our final indexing scheme we reviewed is based on a peer 2 peer(P2P) network called the P-tree designed by Crainiceanu et al [11]. We descided to look at this scheme especially since P2P networks is an upcoming and exciting area to investigate which allows us to view and analyze our data at a different perspective. The following sections describe the general structure and algorithms for each surveyed indexing scheme.

4.2.2 PR-tree

A disadvantage to the R-tree is when performing worst case queries. These result in creating a window covering the whole tree. Arge et al has improved R-tree to account for these instances called the PR-Tree [7]. The PR-tree is generated from a Pseudo PR-tree. The Pseudo PR-tree is different from the R-Tree where each node contains four additional items.

Each node contains the Xmin, Xmax, Ymin, and Ymax. These values are considred the extreme values within a data set. By using the structure of a Pseudo PR-tree, they can take the advantage and convert this in a PR-tree that can reduce the coverage areas in the bounding box for worst case scenrios. This process is very convoluted and not well explained within their paper, thus we could not implement the full conversion from a Pseudo PR-tree. However, this is very beneficial for our query engine, since multiple video sets are used as a single data set to be queried upon.

Index Structure

Based on the clustering data set scheme, a single attribute is needed to index each cluster. Thus, we investigated a Pseudo PR-tree described by Arge et al [7]. A Pseudo PR-tree is basically a 2-D k-d tree where multidimensional data cannot be stored like a R-tree. Although, the PR-tree is similar to the R-tree where each node contains a minimal bounding box and the leaves contain the rectangles for the original data set. However, the leaves are not placed in the same level as in R-tree, but at multiple levels for each node.

Since the Pseudo PR-tree does not handle multidimensional data like the R-tree, it will perform significantly faster. For an R-tree, it must accomodate the data to handle multidimensialty where in the pseudo PR-tree it will handle linear data. Based on our experiments, the query execution time will perform better than the R-tree.

Each node in the Pseudo PR-tree contains six different values. A four dimensional point $((x_{min}, y_{min}), (x_{max}, y_{max}))$ which is based on the extreme points of the original data set. The extreme points are based on the data set. In our data set, since we are indexing based on a single attribute, the root node will contain the lowest value as the x_{min} and y_{min} and the maximum value as the x_{max} and y_{max}. If there is more data between the min and max then a binary split is performed where each division contains the most minimal amount. The division point is considered the T_{min} where T is the current node. Each division will contain another recursive Pseudo PR-tree with six values.

In our data set, in order to obtain the extreme points for a single attribute. The values for a single attribute was sorted based on how extreme the data was. For example, the feature values for 10 images on a single attribute range from 1 through 10. This set will be sorted in the form of $(1, 10, 2, 9, 3, 8, 4, 7, 5, 6)$ where the root node will contain the priority leaves of 1 as the x_{min} and y_{min}, and the value 10 as the x_{max} and y_{max}. Figure 4.1 shows the structure of each node in the pseudo PR-tree.

Below is the algorithm to construct a pseudo PR-tree with a set of 8-feature images. The load algorithm will recursively build the sub Pseudo PR-trees. According to Arge et al [7], this construction time will take $O(\frac{N}{B} log_{M/B} \frac{N}{B})$ I/Os for a set of N input rectangles.

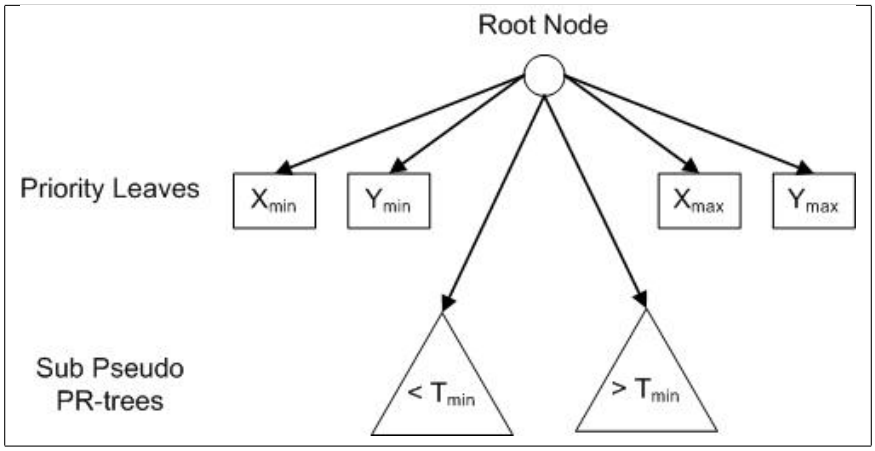


Figure 4.1: A Pseudo PR-tree where the priority leaves contain the extreme range for $((x_{min}, y_{min}), (x_{max}, y_{max}))$ and two recursively sub Pseudo PR-trees.

Algorithm: Pseudo PR-tree Construction
INPUT: 8 Feature Image Data Set
OUTPUT: Pseudo PR-tree

0: Calculate the extreme ranges for an attribute of the data set
1: Store the $((x_{min}, y_{min}), (x_{max}, y_{max}))$ as the extreme range
2: If the number of data points within extreme range is more than one
3: Calculate the binary split as T_{min}
4: For the values $<$ than T_{min}, go to step 1
5: For the values $>$ than T_{min}, go to step 1
6: Else if the number of data points is 1, create leaf for point

Once the Pseudo PR-tree is created, it can then be converted into a PR-tree described by Arge et al.

Search

The search within a PR-tree will perform faster than a R-tree since the the extreme data points are generated by the Pseudo PR-tree and the rectangular bounds are minimized when the PR-tree is created. The query will be very similar to a R-tree, but the bounding box regions will defined differently. In a R-tree, it needs to traverse through the whole tree and may need to go through different parts of the tree based on the range query, thus being slower than a Pseudo PR-tree. The search algorithm for the PR-tree will initially check if the query attribute value is within range of the extreme. If its true, then it will add these leaves into a data structure since these can potentially be the closest image to the query image. Then it would traverse to a sub-tree and keep going through this same process until there is no more children left. According to Arge et al, the query takes $O(\sqrt{N/B} + T/B)\ I/Os$.

Insertion and Deletion

The insertion and deletion is very similar to a R-tree since the general structure of a PR-tree is similar. Arge et al have developed a bulk loading technique for large amounts of insertions. Since numerous insertions will affect the extreme ranges found in the initial stage of creating the Pseudo PR-tree.

4.2.3 Indexing of Variable Length Multi-Attribute Data

Our system contains a multivariate attribute feature set which is difficult to build a single index. Li et al have developed a method to decompose a large variable length attribute set using a Singulare Value Decomposition (SVD) to a relatively smaller set, but with most of the variability still intact [22]. Their main goal is to remove irrelevant data found in 3D motion data. There were able to achieve to prune about 90 percent of irrelevant data. Since our data set is based on motion features, this algorithm should prune out any noise produced by the novelty detection system and produce similar results.

Singular Value Decomposition Basics

The goal of SVD is to reduce a data set with a large amount of values to a smaller set. For any real matrix with m rows and n columns where $m \geq n$, the theorem states

$$\mathbf{A_{mxn}} = \mathbf{U_{mxm}}\mathbf{\Sigma_{mxn}}\mathbf{V^T_{nxn}} \tag{4.1}$$

where U consist of left singular vectors on its column, Σ has singular values on its diagonal, and V has right singular vectors on its rows. U consists of the eigenvalues and eigenvectors

of AA^T and V consists of A^TA eigenvalues and eigenvectors. Σ contains singular values in descending order.

U is initally computed by multiplying the original matrix A with its transpose A^T which we will denote as A'. A' will be in the form of a square matrix of $n\ x\ n$. We can then compute the eigenvalues for A'. For some vector x that is nonzero, x is the eigenvector of A' if:

$$\mathbf{A'x} = \lambda\mathbf{x} \tag{4.2}$$

where λ is considered the eigenvalue for A' and x to be the eigenvector based on λ. The eigenvalue λ can be found first simplifying 4.2 to:

$$(\mathbf{A'} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0} \tag{4.3}$$

where $\mathbf{I}$ is the identity matrix of λ.

Once 4.3 has been found, the determinant of 4.3 needs to be calculated. The $det(A' - \lambda I)$ is found by:

$$\mathbf{\Sigma_{j=1}^{n}(A' - \lambda I)_{ij}C_{ij}} \tag{4.4}$$

where i is the row and j is the column. C_{ij} is found by $(-1)^{i+j}$. This can be further reduced by multiplying each instance of 4.4 by the matrix of $det(ith\ row\ and\ jth\ column\ removed)$ if the size of the matrix is greater than 3. Once 4.4 has been reduced, it will become a n degree polynomial. The nonzero distinct factors for the polynomial are the eigenvalues for 4.3.

Eigenvectors are found by substituting the eigenvalues into 4.3 and solving the system. The eigenvectors are combined into a column of a matrix which will result into the left singular vector U.

The right singular vector V is found similarily as U by multiplying the transpose A^T with the original matrix A. Since matrix multiplication is not cummative, this will result into a different vector.

The singular vector Σ is found by the square root of either eigenvalues for AA^T of U or A^TA of V. They are then placed into a $m\ x\ n$ matrix diagonally in descending order.

Index Structure

This index is based on a Suffix Tree with eight levels for each attribute in the data set which is based on the values generated by SVD for the left singular vectors U. Each node is separated into three intervals where each interval is one third of the max of the computed U matrix for a single attribute. Figure 4.2 shows the structure of this index along with an example query. For example, if the range is from 0.0 to 1.5 for the first attribute, there will be three intervals of $0 - 0.5,\ 0.5 - 1.0$, and $1.0 - 1.5$. Like Li et al, we have also negated our negative values for the U matrix before inserting them into the index.

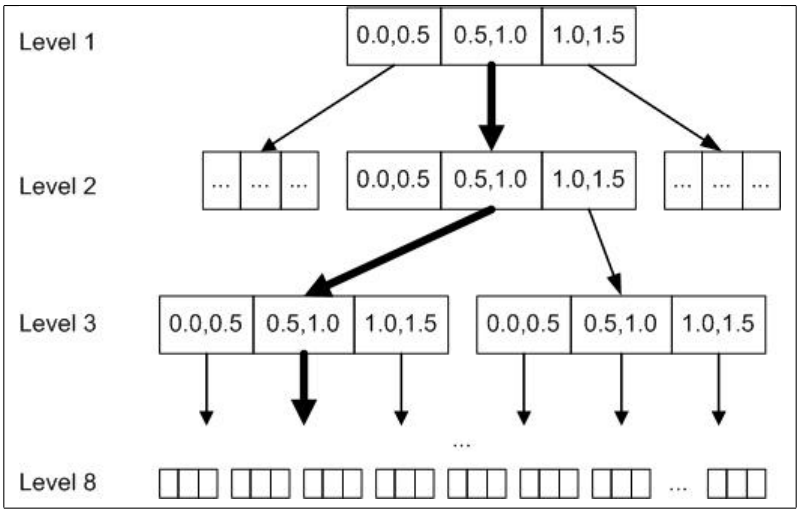


Figure 4.2: A SVD Index Tree with three intervals per node and an example on bold lines.

An advantage to this indexing scheme over R-tree is an approximate indexing is not needed. All the attributes are represented as each level within the index tree. An n dimensional data set can be used in this index and maintain performance where an R-tree will worsen as n increases.

It is possible to have more than three intervals for each node. In Li et al's design, they used four intervals for each node. We believed that by having too many intervals within each node, it would have pruned most of the similar images within the index. The retreival image would only be an exact match, since the images will be very spread apart based on having a large interval set.

Search

For every image to be queried, the left singular vector U from the SVD must be calculated. Each calculated attribute value will represent each level within the index. For both the unclustered and clustered data sets, a single root node is looked at for the query. Below is the algorithm for the query:

Figure 4.2 shows an example if the query image contained the SVD left singular vector values of (.6,.8,.2,.8,1.2,1.3,1.1,.2). Level 8 contains the image information which will return to the requester.

Insertion and Deletion

This paper does not provide any information on how to insert or delete from this index since its very expensive to recalculate the SVD for the whole data set. It is necessary to

Algorithm: Search using SVD
INPUT: Query Image
OUTPUT: Most Similar Retrieval Image

```
0:  Calculate the left singular vector for Query Image
1:  For each feature starting at root node
2:      determine interval path for current feature
3:      traverse to next node
4:  end For until reached level 8
5:  Return Retrieval Image
```

recalculate this data set since each level within the index is associated with the next level. If an insertion or deletion must be done, there are two methods of doing this. The first is that the current index containing the SVD values can converted back to the original matrix A or regenerated from the original data set. Insertion will simply be recalcuating the left SVD by adding the new image to the data set, where deletion can be done by removing it from the data set and then calculating the SVD.

4.2.4 HDoV-tree

An interesting method of viewing a data set uses the HDoV-tree designed by Shou et al [34]. It is intended to view 3D objects based on a point of view. For example, if you were standing in front of a set buildings, perhaps at a school campus or a small city. The closer buildings to you will have more detail, where buildings further away will have a lower amount of detail. This concept is based on one of two principles called the Level of Detail (LoD). In our example, they believe that closer building should have a higher level of detail since you can see them better than buildings farther away. Also, suppose there was a smaller building behind a larger building at your point of view. The smaller building will not be visible to you and thus not counted as part of your visibility. This is the second concept of the HDoV-tree called the Degree of Visibility (Dov). It is based on a perspective where some objects may not be as visible as other objects in front or larger than them.

Degree of Visibility Structure

Our data set is a set of novel images from a video stream. Eventhough our data set is not of the 3D form, we can utilize these two concepts of detail and visibility to our R-tree design. Shou et al [34] measures the degree of visibility as a value between 0 and 1 where the internal node is the total sum of the DoV of all its children. In our system, our main goal is to find the most similar image(s) within the novelty data set. We believe the DoV is how similar the query image is with every other image within the data set. This will provide us more information about the data set when we perform our queries.

The similarity between two images is measured by the Euclidean distance. The reciprocal of this distance will provide us a value between 0 and 1 for the DoV. This is similar to the

Algorithm: Degree of Visibility Creation
INPUT: R-tree of data set
OUTPUT: Calculated DoV for each node

0: Traverse to last level of R-tree
1: If reached a Leaf
2: Sum together distances between Query Image and each image in Leaf
3: Else,
4: Sum together DoV for each Node
5: Set the DoV to the Sum
6: If node is not the root node, go to step 1

values Shou et al have used for their HDov-tree. Since the DoV is based on the query image or point of view, this is considred dynamic since these values are change for every new query, insert and delete. In order to save processing time on the DoV for each query, we use the same concept as the "dirty bit". When consecutive inserts or deletes occur within the data set, the node is considered dirty. This will allow the DoV not to be recalcuated for every insert or delete. This bit will only be cleaned when a traversal occurs through this node. However, this will add to the retrieval time for this recalculation. For every query, the DoV will be created for each node within the tree. In the DoV Creation algorith, it will initially traverse to the lowest level containing the leaves within the R-tree. It will then recursively go up through the tree while assigning the DoV for each node until it reaches the root node.

Level of Detail Structure

In our example above, buildings with a higher degree of detail should be displayed with a higher resolution. When buildings are farther away, we do not need to see the full detail of the image to get an idea of what it represents which will be displayed at a lower resolution quality. In our data set, the amount of resolution is not needed since the distance between images does not define the actual physical distance like HDoV trees. But we can incorporate this idea within the tree structure by approximating the actual image content without looking at the data itself. We use the idea from clustering to get the best representative image for a set of children under a single node. This will occur recursively up to the root of the tree. Then for each node within the tree, it will be the best approximation for the images below which will provide us with the Level of Detail each sub-tree pertains to. The LoD will not change as often as the DoV, since its independent from the query image. It is only necessary to modifiy the LoD when it has been deleted from the data set. The Level of Detail Creation Algorithm is similar to the DoV Creation algorithm in where it also starts from the bottom level of the tree and recursively works its way to the root. In each subtree, the algorithm will calculate the average feature value. This algorithm is using the approximate indexing scheme on a single algorithm. Based on the average, this algorithm will find the closest

Algorithm: Level of Detail Creation
INPUT: R-tree of data set
OUTPUT: Calculated LoD for each node

0: Traverse to last level of R-tree
1: If reached a Leaf
2: Sum together Image features for each image
3: Else,
4: Sum together LoD for each Node
5: Find average of the set of images
5: Set the LoD to node closest to average
6: If node is not the root node, go to step 1

Algorithm: HDoV-tree Search
INPUT: Query Image
OUTPUT: Most Similar Retrieval Image

0: For each node starting at the root node
1: If the DoV is greater than the Threshold
2: If the node is a leaf, then return the images in Node
3: Else, go to step 1
4: Else, return the LoD of Node

image within the current set and it will be considered the best representative image for a subtree.

Search

The greatest utility of the HDoV tree is its performance on the level of quality retrieved. Considering 3D applications such as Graphical Information Systems (GIS) which require a data to be returned, this indexing scheme concentrates on the types of data that really matters and pay less attention to those not visible to a point of view. Based on our interpretation of the HDoV tree, the further we traverse the tree, the less likely we will retrieve something relevant. Since each node contains a DoV which tells the system how similar each subtree images are to the query image. Based on a certain pre-defined threshold, we can determine how far we want to travel through the tree. Once we have reached our optimal threshold and since the LoD is already specified within the node, we can return this image without looking at the leaf nodes. The search algorithm for the HDoV-tree initially starts at the root of the tree. From each node, the algorithm will compare the node's DoV with the threshold. If this is true, it will keep traversing through the tree and comparing it to the threshold until a leaf node is reached. When the DoV of a certain node is less than the threshold, then the LoD image will be returned. This is the beauty of this indexing scheme, since the best representative image is already calculated for this subtree. Its not necessary to proceed to the rest of the nodes. This will save in the retrieval time of a query.

Insertion and Deletion

When a new image is inserted into this index, the process will be very similar to a R-tree. The only difference is the set of a dirty bit discussed in the previous sections. This dirty bit will be a flag to the the search algorithm that this subtree needs to be recalculated. However, when an image is deleted, our system will need to rebuild the HDoV tree since the LoD nodes are not consistant with all the other values.

4.2.5 P-tree

Since peer to peer (P2P) networks have been very popular lately, Crainiceanu et al has developed a P-tree index that is based on P2P networks [11]. This indexing scheme provides an efficient distributed network that is fault tolerant and able to support range queries. This tree's structure is in the form of a circle or a ring where each peer is connected together throughout the network. A single value is used to represent each peer within the circle, where each peer thinks that this value is the lowest of all the values. P-trees work well in a distributed network since each peer does not hold the entire index for the data set.

Everytime a new peer becomes part of the network, there are two mail algorithms. The first algorithm is called the ping process where each peer in the network periodically checks every other peer for uniformity. When a peer leaves the network, during the ping process it will identify the leaving peer as being not uniform and will not satisfy the coverage and seperation properties. The data within each B-tree will become inconsistant from this peer departure and will be fixed using the stabilization process.

Index Structure

The P-tree is based on a B+tree like structure. However, there are two main differences in a P-tree. The representative value for each peer is located at the leftmost node of the tree on the first level. The leftmost values pertain to the P-tree for that node and the other values are meant for other peers within the network. There can be multiple levels of the P-tree based on the amount of peers within the network. The number of nodes at each level is based on the order d specified before the P-tree creation. The first level of the P-tree can contain between 2 to $2d$ nodes where the children can have from d to $2d$ nodes.

Initially when the network is created and a peer is added, P-tree's two algorithms peer and stabilization is executed. When each peer is added to the network, the peer and stabilization executions is not needed but will improve the search time. Each peer's P-tree is restructured whenever a new peer is added to the network. Figure 4.3 shows the overall structure along with an example.

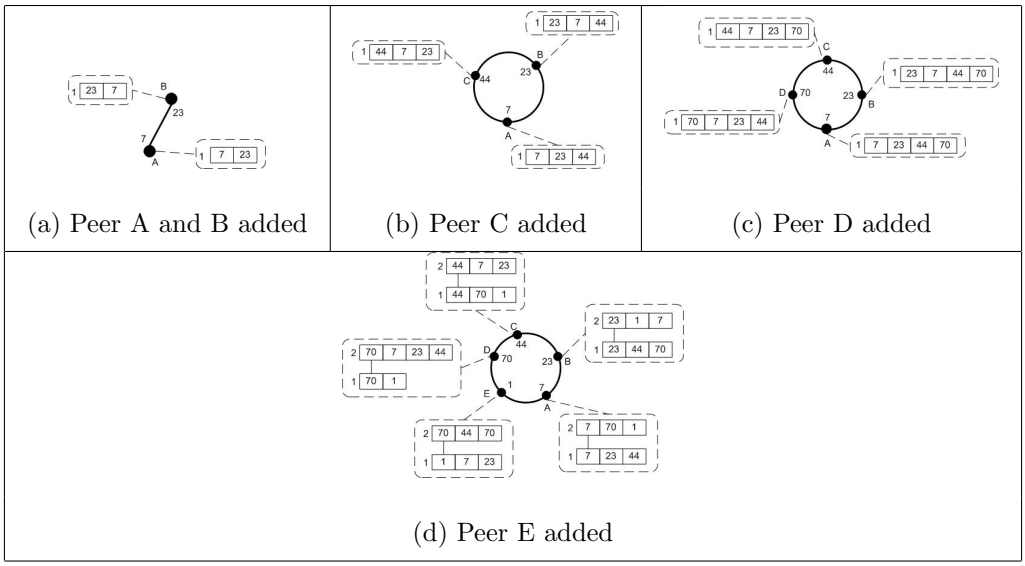


Figure 4.3: The process of building a full P-tree.

Algorithm: Initial P-tree Construction
INPUT: Peers 1 thru 4
OUTPUT: 4 Peer P-tree

0: If total peers in P-tree is four then
1: Find previous occurance of node
2: Insert new Peer value previous node occurance
3: Else, Go to P-tree New Level Creation

In our implementation, we have designated an order of d of 2. This means that at any level, there cannot be more than four nodes. The example in Figure 4.3 is an example how a P-tree is built. Initially for the first peers that enter the network (Figure 4.3a-c), they all belong to the same level. They are sorted based on the position within the ring. The important aspect of this to remember is the actual peer value is not used to determine if one peer is less than another peer. A peer is less based on if the peer came into the network before the other peer. This structuring process is described in the Initial P-tree construction algorithm. Once peer five have been added to the network, a new level is added. In a P-tree, the root node represents the highest level. In Figure 4.3d, the root is on level 2. The node that is place on the leaf node within Figure 4.3d is based again on the order precedence. On node D for this figure, the peer 1 has been dropped down to the leaf node because its "less" than the second node at the root level. This is applied to each peer. For peers more than four in the P-tree is explained in the P-tree New Level Creation. Once the initial construction has been completed, there is a second phase of construction not depicted in Figure 4.3d. This has to do with our assumptions on the algorithm since the original design was very convoluted and hard to understand. Our first assumption is based on overlap between two peers for coverage of the data set. For the second node at the root level, this

Algorithm: P-tree New Level Creation
INPUT: Peer Node
OUTPUT: P-tree New Level

```
0:  If new peer is less than the second peer
1:      add peer to leaf node
2:  Else,
3:      replace second node at root level in leaf level
4:      place original second node at leaf node
5:  Go to P-tree Regeneration
```

Algorithm: P-tree Regeneration
INPUT: Inital Construction of P-tree
OUTPUT: Full Construction of P-tree

```
0:  For each peer p in the P-tree
1:      Contact the second peer at root level
2:      Obtain the rightmost node at leaf level
3:      If obtained node value is different from peer
    p's value and not in peer p's leaf nodes then
4:          Insert into root level node
5:          If root level overflows, then create new level
    and insert highest value
6:          Contact inserted node
7:          Go to Step 2
8:      Else, done with this peer
9:  End For when visited each peer in the network
```

node has to be traversed in its leaf level. For the last node in the leaf (bottom) level needs to be referenced from the initial node. This process continues until a node is found within the original peer or if the value is the peer itself. If there is no more room in either of the two levels, then a third level is created. The highest value node is placed on the third level. This level would then be considered as the root level. This is called a regeneration process which is explain in the P-tree Regeneration algorithm. This algorithm is only required if there are more than four peers in the network. When a peer leaves the network, each peer needs to remove its pointer to that peer. Then once all the peers have removed that peer, then the P-tree Regeneration needs to be called again. Also, its only called if the resulting network is greater than four nodes.

Search

Searching for an image using the P-tree is very interesting, since there are two indexing schemes required. The first is the P-tree, then the R-tree index stored at each Peer. Any peer within the network can execute a search query. Initially the peer searches within its own leaf nodes for the closest match. If its found within the leaf nodes, then it contacts that peer value with the query. The peer then searches its own R-tree and retrieves the image and sent back to the requesting peer. If the image was not found within the leaf node

Algorithm: P-tree Search
INPUT: Query Image on requesting Peer
OUTPUT: Retrieval Image

```
0:  If Image value is closest to current peer value
1:      search own R-tree for value
2:      retrieve image and send back to requester
2:  Else, If image is in leaf level
3:      Contact Peer and Send Image to Peer
4:      Contacted Peer will go to step 0
5:  Else, If image not in leaf level, search other levels
    for closest match
6:      Contact Peer and Send Image to Peer
7:      Contacted Peer will go to step 0
```

level, then it would go through the levels other than the leaf level until it finds the closest match and contacts the node. The complexity of searching a P-tree for the correct peer is $O(Log_d N)$ where d is the order of the P-tree. However, if the P-tree changes because of new and old peers very often, the complexity increases which results close to O(N). This process is described in the P-tree Search Algorithm.

Insertion and Deletion

The insert and delete process is very similar to the query to find the image first. Since each peer represents the represented image for each cluster, the removal or insert may change the represented image. In order to conserve to restructure all the P-trees, we have set a threshold where if the insert or delete affects the centroid by a certain threshold, then recalculate representative image and regenerate P-trees.

4.3 Experiments and Results

Our results are based on six real world stationary surveillance videos. These videos was taken from office buildings and school campuses. Our novelty detection system detected an average of 100 novel key-frames and 250 regions from each video. We performed 10 fold testing for each fold consisted of a 90-10 split. Our metrics is based on distance, time, and quality. Distance is based on the Euclidean distance between the query and retrieved image. Time is the number of mili-seconds it took to retrieve the image. All of our tests were performed on the same system to ensure consistant times. Quality is based on a manual observation rating of 0 or 1. Images that were similar had a value of 1 and otherwise a 0.

4.3.1 Video Result Details

We executed experiments with the image query engine using our clustering algorithm on six videos seperatly and together. We compared these results with the same image query

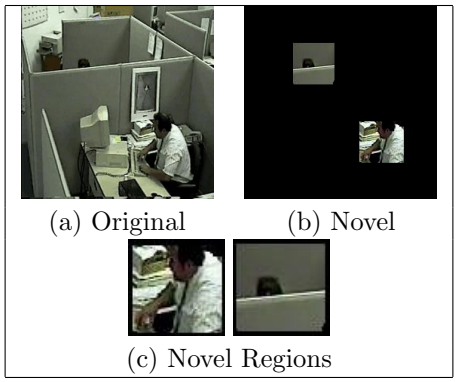
	Non-Clustered			Clustered		
Folds	**Distance**	**Time**	**Quality**	**Distance**	**Time**	**Quality**
1	4156.42	6.51	0.11	1799.00	2.32	0.41
2	3309.98	5.12	0.19	1453.00	3.02	0.42
3	3681.70	6.53	0.23	1752.00	3.02	0.47
4	3227.56	6.05	0.09	1510.00	3.26	0.42
5	2715.70	6.09	0.16	1651.00	2.09	0.40
6	4708.91	5.37	0.09	2422.00	3.02	0.35
7	3318.98	4.93	0.16	1787.00	3.05	0.37
8	4479.56	5.35	0.12	2281.00	3.02	0.40
9	4152.44	5.58	0.12	1782.00	3.26	0.35
10	3476.72	4.42	0.19	2180.00	2.58	0.47
Avg.	**3722.80**	**5.60**	**0.15**	**1861.70**	**2.87**	**0.40**

Table 4.1: Video 1: 10 fold Non-Clustered vs. Clustered Region Raw Data

engine, but without clustering the six videos. These experiments were performed on novel key frames and regions for each video. Our expected results for each video was to show that by using our clustering index performed better than our non-clustered index in terms of distance, time, and quality.

Video 1 Results

The first video is based on an office environment of where a man is sitting at his computer. After some time, the man at the desk gets frustrated and knocks the computer off the desk. Another person in the next cubicle looks over to see what the commotion is about. Figure 4.4 shows an image taken from this video (a), the novel image generated by VENUS (b), and the extracted regions for this image (c). This scene shows the man pounding at his



(a) Original (b) Novel

(c) Novel Regions

Figure 4.4: Video 1: Original, Novel, and Novel Regions (Not Actual Size)

computer, while person in the next cubicle begins to stand up.

In Figure 4.5 is a graphical representation of Table 4.1 which shows the 10 fold testing that was performed to test between clustered and non-clustered data for region data. Our expected results were correct where the clustered data set performed better than the non-clustered set. The distance for the clustered set is smaller than the non-clustered, thus showing a better degree of quality. Also the retrieval time was less for the clustered set.

We also performed the same type of tests between clustering and non-clustering with non-region images. Non-region images refer to the novel images generated by the VENUS system. Figure 4.6 is a graphical representation of Table 4.2.

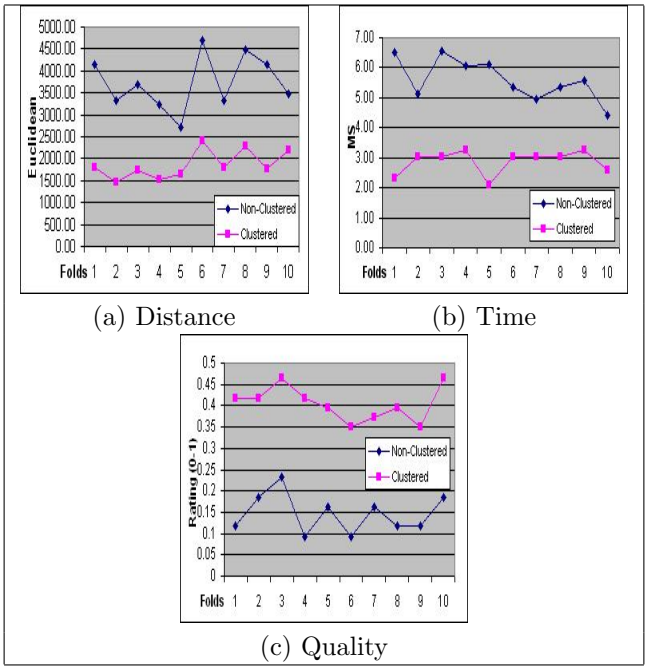
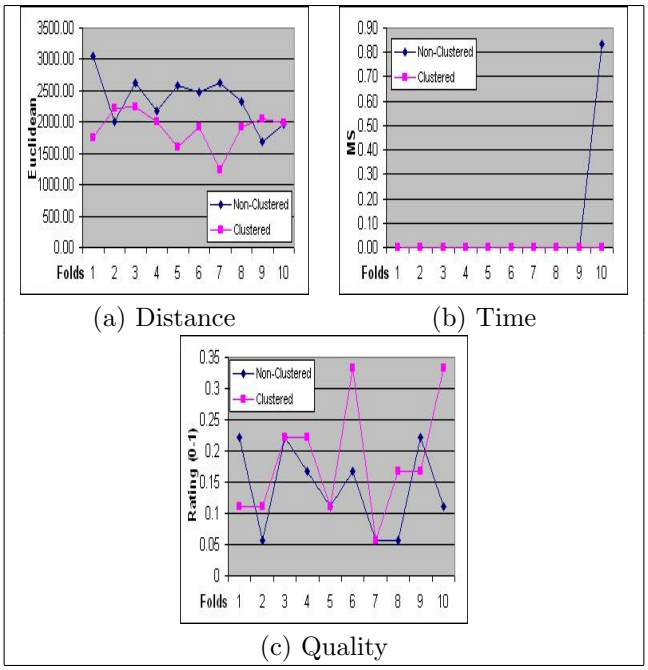


Figure 4.5: Video 1: 10 fold Non-Clustered vs. Clustered Regions

	Non-Clustered			Clustered		
Folds	**Distance**	**Time**	**Quality**	**Distance**	**Time**	**Quality**
1	3052.06	0.00	0.22	1755.17	0.00	0.11
2	2014.22	0.00	0.06	2225.83	0.00	0.11
3	2624.50	0.00	0.22	2232.22	0.00	0.22
4	2172.22	0.00	0.17	2004.89	0.00	0.22
5	2573.11	0.00	0.11	1600.89	0.00	0.11
6	2480.00	0.00	0.17	1913.17	0.00	0.33
7	2633.22	0.00	0.06	1228.17	0.00	0.06
8	2318.06	0.00	0.06	1913.72	0.00	0.17
9	1677.78	0.00	0.22	2050.94	0.00	0.17
10	1954.17	0.83	0.11	1982.50	0.00	0.33
Avg.	**2349.93**	**0.08**	**0.14**	**1890.75**	**0.00**	**0.18**

Table 4.2: Video 1: 10 fold Non-Clustered vs. Clustered Non-region Raw Data



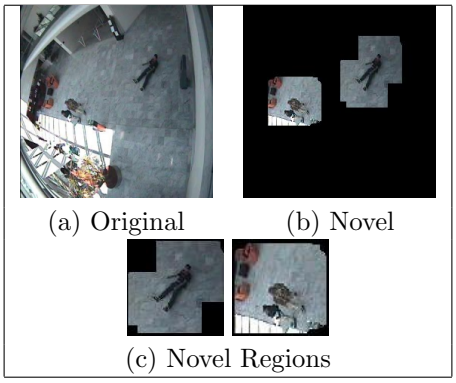
(a) Distance (b) Time

(c) Quality

Figure 4.6: Video 1: 10 fold Non-Clustered vs. Clustered Non-regions

Video 2 Results

The second video is located inside of a building where a fight occurs. Figure 4.7 shows an image taken from this video (a), the novel image generated by VENUS (b), and the extracted regions for this image (c). This scene shows a man laying on the ground after a



(a) Original (b) Novel

(c) Novel Regions

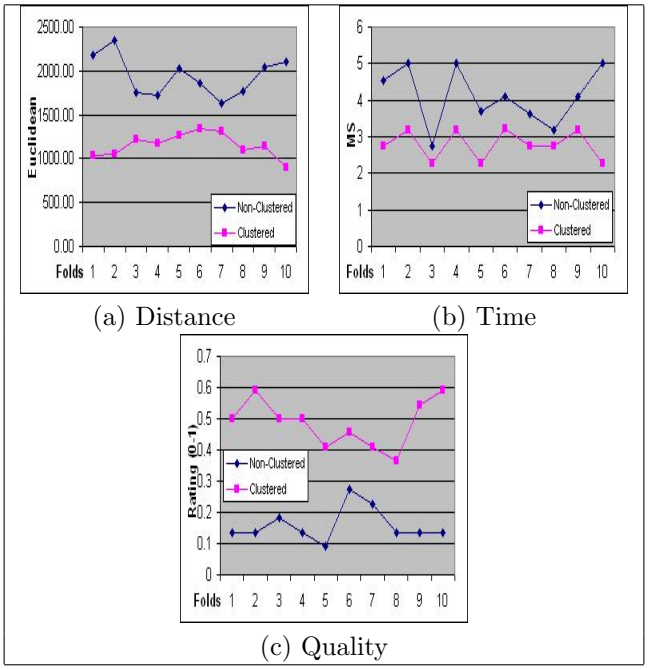
Figure 4.7: Video 2: Original, Novel, and Novel Regions (Not Actual Size)

fight on the right and two persons standing on the left.

In Figure 4.8 is a graphical representation of Table 4.3 which shows the 10 fold testing that was performed to test between clustered and non-clustered data for region data. These results are as expected and similar to the previous video. The distance for the clustered set is smaller than the non-clustered, thus showing a better degree of quality. Also the retrieval time was less for the clustered set. We also performed the same type of tests between

	Non-Clustered			Clustered		
Folds	Distance	Time	Quality	Distance	Time	Quality
1	2186.50	4.55	0.14	1034.00	2.73	0.50
2	2349.05	5.00	0.14	1059.00	3.18	0.59
3	1758.95	2.73	0.18	1223.00	2.27	0.50
4	1723.36	5.00	0.14	1170.00	3.18	0.50
5	2023.14	3.68	0.09	1262.00	2.27	0.41
6	1862.45	4.09	0.27	1338.00	3.23	0.45
7	1628.82	3.64	0.23	1318.00	2.73	0.41
8	1764.64	3.18	0.14	1102.00	2.73	0.36
9	2039.86	4.09	0.14	1148.00	3.18	0.55
10	2104.59	5.00	0.14	901.00	2.27	0.59
Avg.	**1944.14**	**4.10**	**0.16**	**1155.50**	**2.78**	**0.49**

Table 4.3: Video 2: 10 fold Non-Clustered vs. Clustered Region Raw Data



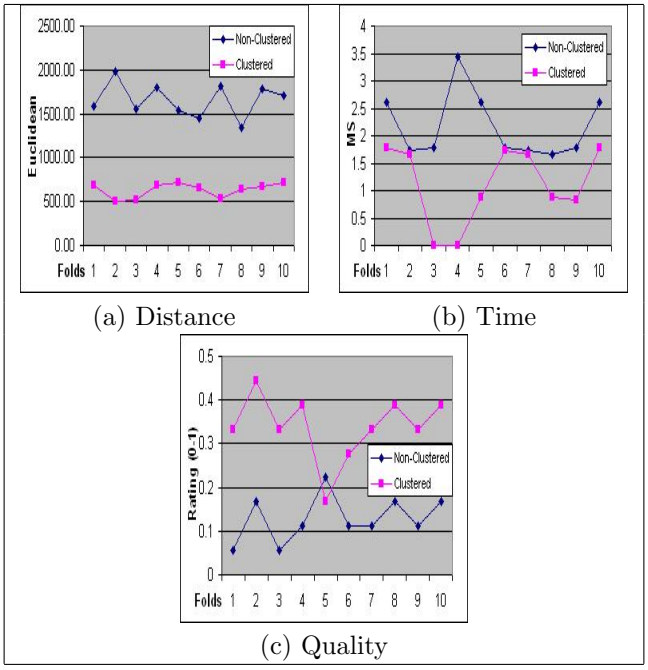
(a) Distance (b) Time

(c) Quality

Figure 4.8: Video 2: 10 fold Non-Clustered vs. Clustered Regions

	Non-Clustered			Clustered		
Folds	**Distance**	**Time**	**Quality**	**Distance**	**Time**	**Quality**
1	1585.33	2.61	0.06	680.94	1.78	0.33
2	1980.61	1.72	0.17	503.44	1.67	0.44
3	1552.72	1.78	0.06	520.17	0.00	0.33
4	1791.44	3.44	0.11	684.72	0.00	0.39
5	1533.67	2.61	0.22	709.78	0.89	0.17
6	1449.06	1.78	0.11	658.00	1.72	0.28
7	1806.67	1.72	0.11	532.89	1.67	0.33
8	1339.67	1.67	0.17	639.22	0.89	0.39
9	1788.39	1.78	0.11	664.72	0.83	0.33
10	1711.72	2.61	0.17	710.44	1.78	0.39
Avg.	**1653.93**	**2.17**	**0.13**	**630.43**	**1.12**	**0.34**

Table 4.4: Video 1: 10 fold Non-Clustered vs. Clustered Non-region Raw Data



(a) Distance (b) Time

(c) Quality

Figure 4.9: Video 2: 10 fold Non-Clustered vs. Clustered Non-regions

clustering and non-clustering with non-region images. Non-region images refer to the novel images generated by the VENUS system. Figure 4.9 is a graphical representation of Table 4.4.

Video 3 Results

The third video is located at our laboratory of people walking from right to left. Figure 4.10 shows an image taken from this video (a), the novel image generated by VENUS (b), and the extracted region for this image (c). This scene shows a man walking from right to left.

In Figure 4.11 is a graphical representation of Table 4.5 which shows the 10 fold testing that was performed to test between clustered and non-clustered data for region data. These results are as expected and similar to the previous videos. The distance for the clustered set is smaller than the non-clustered, thus showing a better degree of quality. Also the retrieval time was less for the clustered set. We also performed the same type of tests between

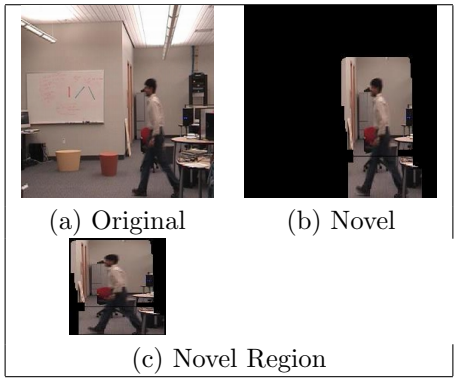


Figure 4.10: Video 3: Original, Novel, and Novel Region (Not Actual Size)

	Non-Clustered			Clustered		
Folds	**Distance**	**Time**	**Quality**	**Distance**	**Time**	**Quality**
1	4482.00	8.00	0.40	994.00	4.00	0.40
2	4338.00	2.00	0.60	1135.00	0.00	0.60
3	4033.00	8.00	0.20	1275.00	0.00	0.60
4	4856.00	2.00	0.40	1397.00	0.00	0.60
5	3947.00	6.20	0.40	1524.00	0.00	0.20
6	3455.00	6.00	0.20	1228.00	0.00	1.00
7	5865.00	6.00	0.00	1169.00	0.00	0.80
8	5772.00	8.00	0.20	1414.00	0.00	0.60
9	4215.00	10.00	0.80	1615.00	2.00	0.60
10	6669.00	6.00	0.40	1288.00	2.00	0.40
Avg.	**4763.20**	**6.22**	**0.36**	**1303.90**	**0.80**	**0.58**

Table 4.5: Video 3: 10 fold Non-Clustered vs. Clustered Region Raw Data

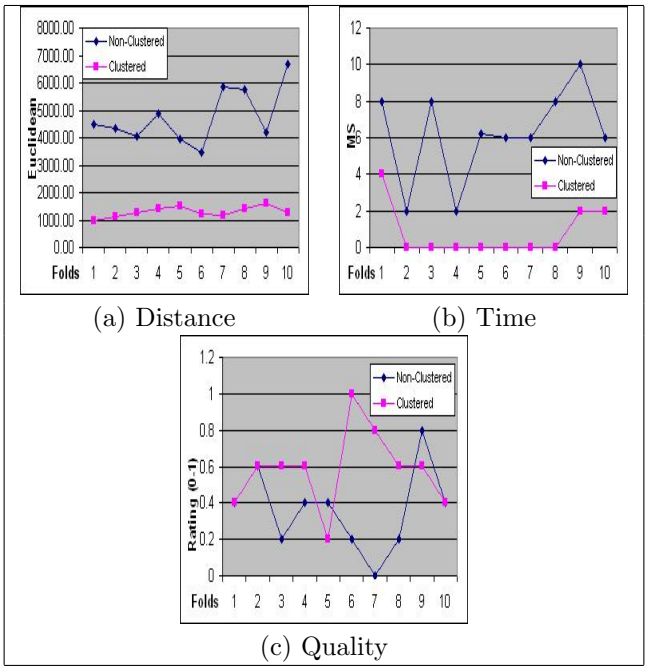
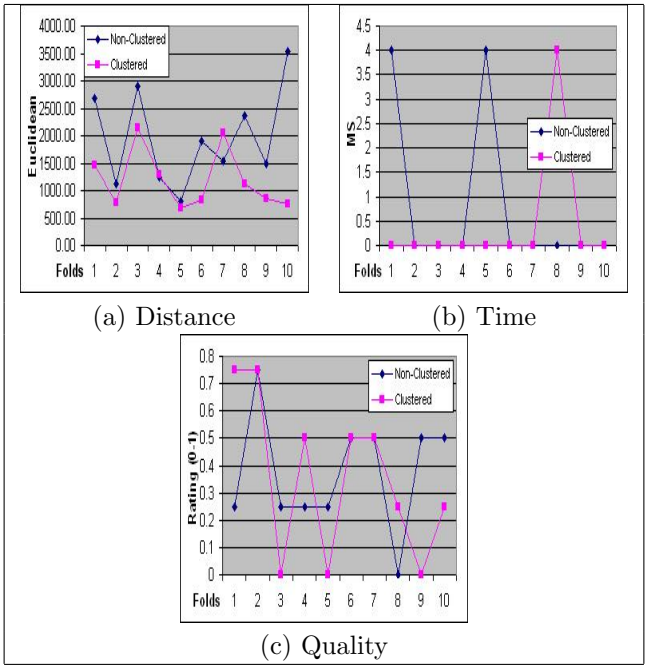


Figure 4.11: Video 3: 10 fold Non-Clustered vs. Clustered Regions

	Non-Clustered			Clustered		
Folds	**Distance**	**Time**	**Quality**	**Distance**	**Time**	**Quality**
1	2693.75	4.00	0.25	1473.50	0.00	0.75
2	1125.00	0.00	0.75	782.25	0.00	0.75
3	2900.25	0.00	0.25	2144.25	0.00	0.00
4	1251.50	0.00	0.25	1295.50	0.00	0.50
5	811.50	4.00	0.25	691.25	0.00	0.00
6	1910.25	0.00	0.50	827.00	0.00	0.50
7	1534.75	0.00	0.50	2039.25	0.00	0.50
8	2362.50	0.00	0.00	1111.25	4.00	0.25
9	1487.00	0.00	0.50	851.25	0.00	0.00
10	3537.25	0.00	0.50	750.75	0.00	0.25
Avg.	**1961.38**	**0.80**	**0.38**	**1196.63**	**0.40**	**0.35**

Table 4.6: Video 3: 10 fold Non-Clustered vs. Clustered Non-region Raw Data



(a) Distance (b) Time

(c) Quality

Figure 4.12: Video 3: 10 fold Non-Clustered vs. Clustered Non-regions

clustering and non-clustering with non-region images. Non-region images refer to the novel images generated by the VENUS system. Figure 4.12 is a graphical representation of Table 4.6.

Video 4 Results

The fourth video is located inside of a building where three people meet together in the middle of the room. Figure 4.13 shows an image taken from this video (a), the novel image generated by VENUS (b), and the extracted regions for this image (c). This scene shows a man laying on the ground after a fight on the right and two persons standing on the left.

In Figure 4.14 is a graphical representation of Table 4.7 which shows the 10 fold testing that was performed to test between clustered and non-clustered data for region data. These results are as expected and similar to the previous videos. The distance for the clustered set is smaller than the non-clustered, thus showing a better degree of quality. Also the retrieval time was less for the clustered set. We also performed the same type of tests between

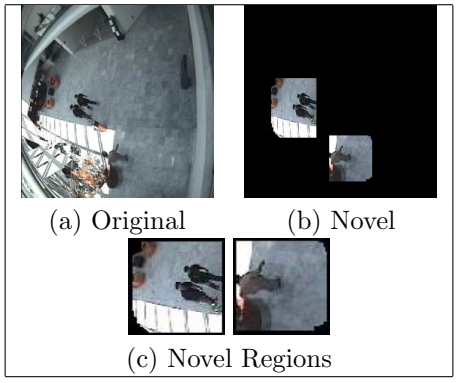


Figure 4.13: Video 4: Original, Novel, and Novel Regions (Not Actual Size)

	Non-Clustered			Clustered		
Folds	**Distance**	**Time**	**Quality**	**Distance**	**Time**	**Quality**
1	3652.67	3.33	0.33	1066.00	0.83	0.92
2	2390.83	5.83	0.33	1040.00	0.00	0.83
3	3133.00	5.00	0.17	1635.00	1.67	0.67
4	1758.00	2.50	0.25	1680.00	0.00	0.58
5	2608.00	3.33	0.25	1116.00	1.67	0.92
6	2810.00	2.50	0.25	1263.00	1.67	0.75
7	2702.00	4.17	0.17	1051.00	0.83	0.67
8	3072.00	3.33	0.25	1355.00	0.83	0.83
9	2801.00	4.17	0.25	1200.00	0.83	0.67
10	2449.00	1.67	0.50	1210.00	0.00	0.75
Avg.	**2737.65**	**3.58**	**0.28**	**1261.60**	**0.83**	**0.76**

Table 4.7: Video 4: 10 fold Non-Clustered vs. Clustered Region Raw Data

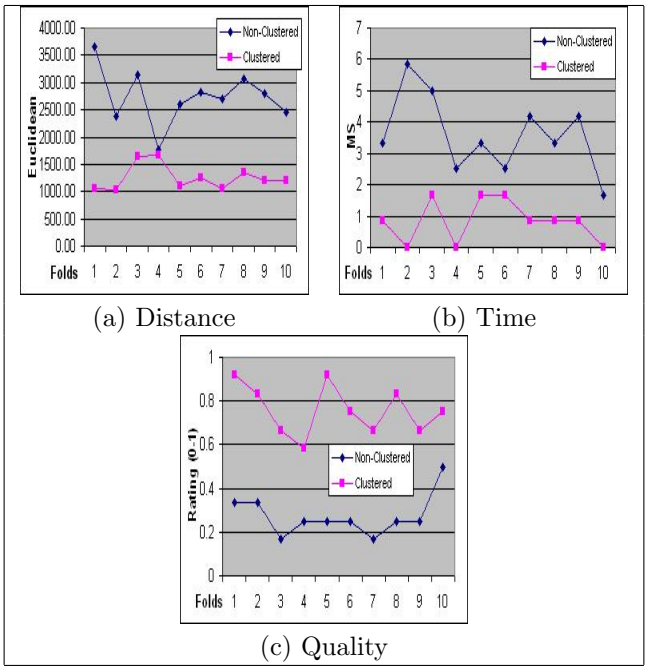


Figure 4.14: Video 4: 10 fold Non-Clustered vs. Clustered Regions

	Non-Clustered			Clustered		
Folds	**Distance**	**Time**	**Quality**	**Distance**	**Time**	**Quality**
1	1947.82	4.36	0.27	1135.00	1.45	0.45
2	1839.82	1.45	0.18	674.36	0.00	0.73
3	3025.09	1.45	0.09	1274.55	1.45	0.45
4	2579.27	2.82	0.18	651.09	1.36	0.45
5	2198.82	0.00	0.27	532.45	1.45	0.91
6	2134.55	1.45	0.18	558.91	0.00	0.64
7	2283.64	0.00	0.18	592.91	0.00	0.36
8	1383.73	5.73	0.18	701.55	2.73	0.55
9	1383.73	2.82	0.09	884.91	0.00	0.55
10	2486.45	2.82	0.27	1057.27	0.00	0.55
Avg.	**2126.29**	**2.29**	**0.19**	**806.30**	**0.85**	**0.56**

Table 4.8: Video 4: 10 fold Non-Clustered vs. Clustered Non-region Raw Data

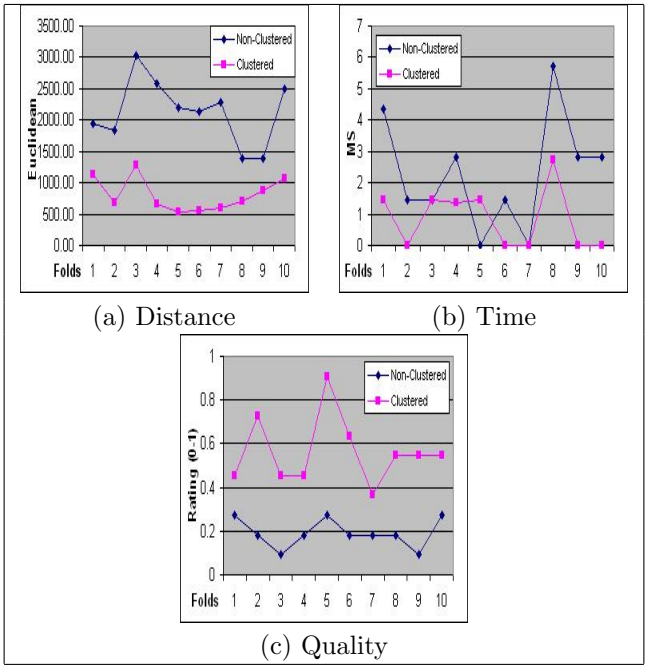


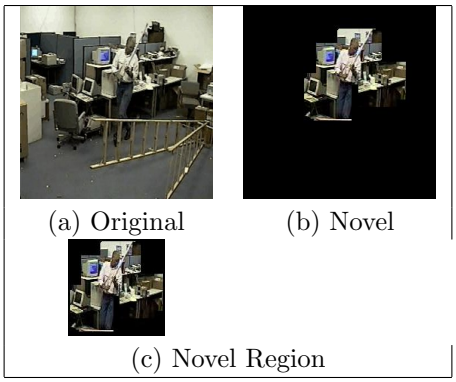
Figure 4.15: Video 4: 10 fold Non-Clustered vs. Clustered Non-regions

clustering and non-clustering with non-region images. Non-region images refer to the novel images generated by the VENUS system. Figure 4.15 is a graphical representation of Table 4.8.

Video 5 Results

The fifth video is about a man trying to receive worker compensation by tipping over a ladder, breaking a light bulb, and then falling to the ground. Figure 4.16 shows an image taken from this video (a), the novel image generated by VENUS (b), and the extracted region for this image (c). This scene shows a man next to a ladder and holding a light bulb. He is about to break the light bulb over the ladder.

In Figure 4.17 is a graphical representation of Table 4.9 which shows the 10 fold testing that was performed to test between clustered and non-clustered data for region data. These results are as expected and similar to the previous videos. The distance for the clustered set is smaller than the non-clustered, thus showing a better degree of quality. Also the retrieval



(a) Original (b) Novel

(c) Novel Region

Figure 4.16: Video 5: Original, Novel, and Novel Region (Not Actual Size)

	Non-Clustered			Clustered		
Folds	**Distance**	**Time**	**Quality**	**Distance**	**Time**	**Quality**
1	2898.00	3.60	0.12	1894.00	1.60	0.52
2	3892.00	3.60	0.16	1569.00	2.40	0.52
3	3956.00	2.80	0.24	1391.00	1.60	0.52
4	2873.00	3.60	0.24	1542.00	2.40	0.52
5	4106.00	4.40	0.32	1705.00	2.80	0.52
6	3453.00	4.84	0.24	1736.00	2.00	0.32
7	4204.00	4.00	0.24	1679.00	2.00	0.40
8	4306.00	3.20	0.20	1995.00	2.00	0.44
9	2949.00	2.80	0.20	1587.00	0.80	0.44
10	3260.00	3.60	0.20	1808.00	1.20	0.56
Avg.	**3589.70**	**3.64**	**0.22**	**1690.60**	**1.88**	**0.48**

Table 4.9: Video 5: 10 fold Non-Clustered vs. Clustered Region Raw Data

time was less for the clustered set. We also performed the same type of tests between clustering and non-clustering with non-region images. Non-region images refer to the novel images generated by the VENUS system. Figure 4.18 is a graphical representation of Table 4.10.

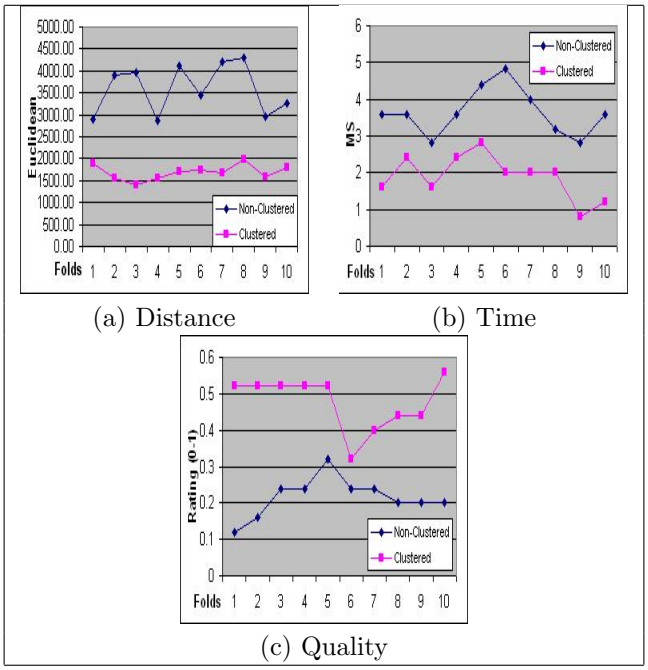
Video 6 Results

The sixth video is located outside on a school campus where students are walking and a white van drives by from left to right. Figure 4.19 shows an image taken from this video (a), the novel image generated by VENUS (b), and the extracted regions for this image (c). This scene shows a white van beginning to enter from the left and a person walking from the right.

In Figure 4.20 is a graphical representation of Table 4.11 which shows the 10 fold testing that was performed to test between clustered and non-clustered data for region data. These

	Non-Clustered			Clustered		
Folds	**Distance**	**Time**	**Quality**	**Distance**	**Time**	**Quality**
1	2850.71	2.82	0.35	1760.12	0.94	0.47
2	3010.35	2.82	0.06	2041.24	1.82	0.18
3	2764.47	2.76	0.41	1435.41	0.00	0.29
4	2103.65	1.76	0.29	1765.71	0.94	0.41
5	1826.59	1.88	0.18	1263.18	2.82	0.18
6	2517.12	0.88	0.06	1150.41	1.76	0.41
7	3216.29	1.88	0.29	2054.12	1.76	0.35
8	2640.94	0.88	0.12	2132.65	0.94	0.12
9	2543.41	2.82	0.18	2353.12	2.76	0.18
10	1913.47	2.71	0.24	1120.12	0.94	0.24
Avg.	**2538.70**	**2.12**	**0.22**	**1707.61**	**1.47**	**0.28**

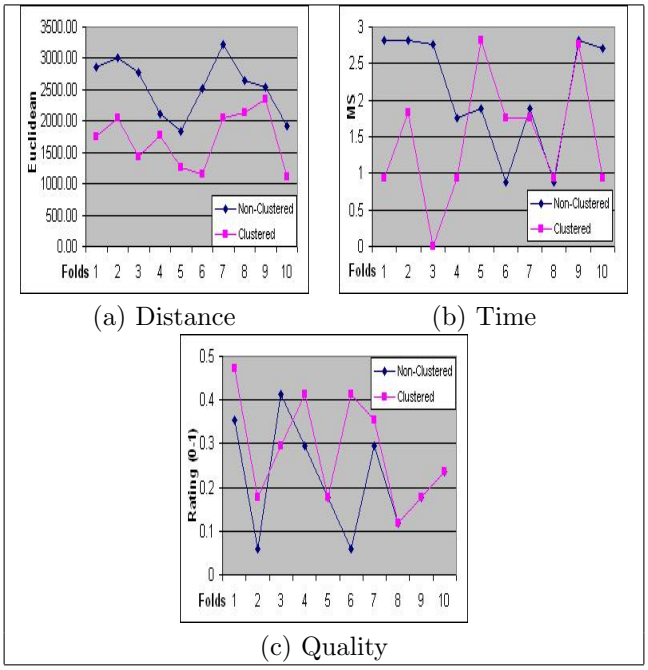
Table 4.10: Video 5: 10 fold Non-Clustered vs. Clustered Non-region Raw Data



(a) Distance (b) Time

(c) Quality

Figure 4.17: Video 5: 10 fold Non-Clustered vs. Clustered Regions



(a) Distance (b) Time

(c) Quality

Figure 4.18: Video 5: 10 fold Non-Clustered vs. Clustered Non-regions

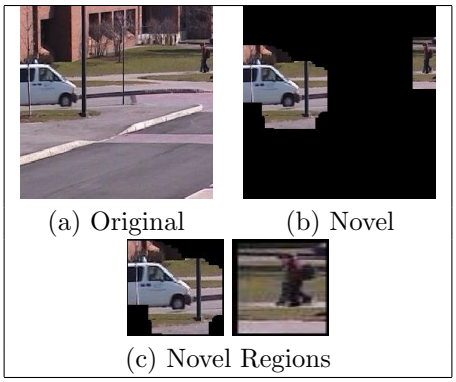


Figure 4.19: Video 6: Original, Novel, and Novel Regions (Not Actual Size)

	Non-Clustered			Clustered		
Folds	**Distance**	**Time**	**Quality**	**Distance**	**Time**	**Quality**
1	2795.00	4.69	0.23	1424.00	2.31	0.38
2	3454.00	3.92	0.23	1709.00	0.00	0.54
3	3794.00	3.85	0.38	1125.00	4.62	0.69
4	3409.00	3.08	0.08	1641.00	3.08	0.62
5	4148.00	3.08	0.08	1184.00	2.31	0.46
6	4341.00	5.38	0.23	2636.00	1.54	0.46
7	4476.00	4.62	0.15	1551.00	0.77	0.46
8	6614.00	5.38	0.23	3223.00	1.54	0.69
9	2908.00	3.08	0.23	1077.00	0.00	0.62
10	3897.00	5.38	0.08	1443.00	1.54	0.31
Avg.	**3983.60**	**4.25**	**0.19**	**1701.30**	**1.77**	**0.52**

Table 4.11: Video 6: 10 fold Non-Clustered vs. Clustered Region Raw Data

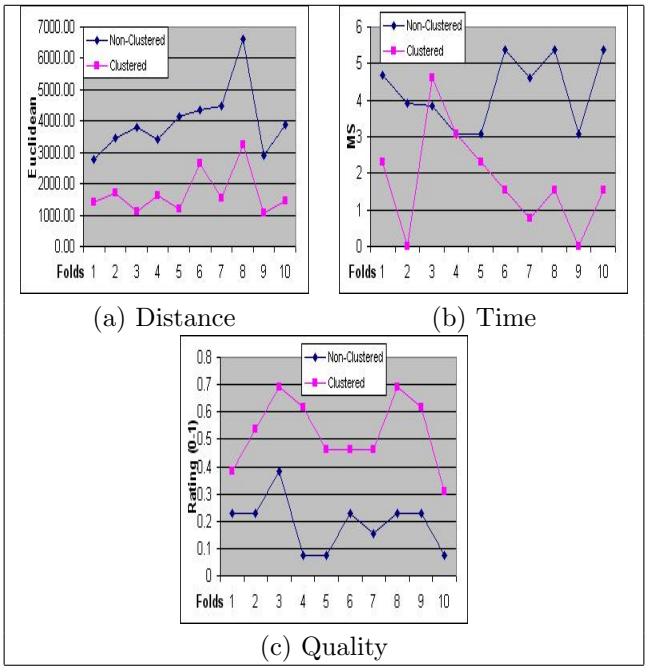
results are as expected and similar to the previous videos. The distance for the clustered set is smaller than the non-clustered, thus showing a better degree of quality. Also the retrieval time was less for the clustered set. We also performed the same type of tests between clustering and non-clustering with non-region images. Non-region images refer to the novel images generated by the VENUS system. Figure 4.21 is a graphical representation of Table 4.12.

All Videos Results

Our final test is to combine all of the six videos into a single data set. This data set was treated the same as the other individual video data sets. In Figure 4.22 is a graphical representation of Table 4.13 which shows the 10 fold testing that was performed to test between clustered and non-clustered data for region data. These results are as expected and similar to all the previous videos. The distance for the clustered set is smaller than the

	Non-Clustered			Clustered		
Folds	**Distance**	**Time**	**Quality**	**Distance**	**Time**	**Quality**
1	3800.80	1.60	0.20	1142.40	0.00	0.50
2	3863.90	3.20	0.10	1129.20	1.50	0.30
3	2280.40	1.60	0.40	875.40	1.60	0.60
4	2420.40	0.00	0.20	787.00	1.50	0.30
5	4141.20	4.70	0.30	3418.90	0.00	0.30
6	4792.20	0.00	0.10	3449.70	0.00	0.40
7	2228.60	1.60	0.20	756.60	1.60	0.30
8	1335.00	3.10	0.30	664.20	1.50	0.50
9	1990.60	1.60	0.20	1040.40	0.00	0.60
10	2253.90	4.80	0.30	990.60	1.60	0.50
Avg.	**2910.70**	**2.22**	**0.23**	**1425.44**	**0.93**	**0.43**

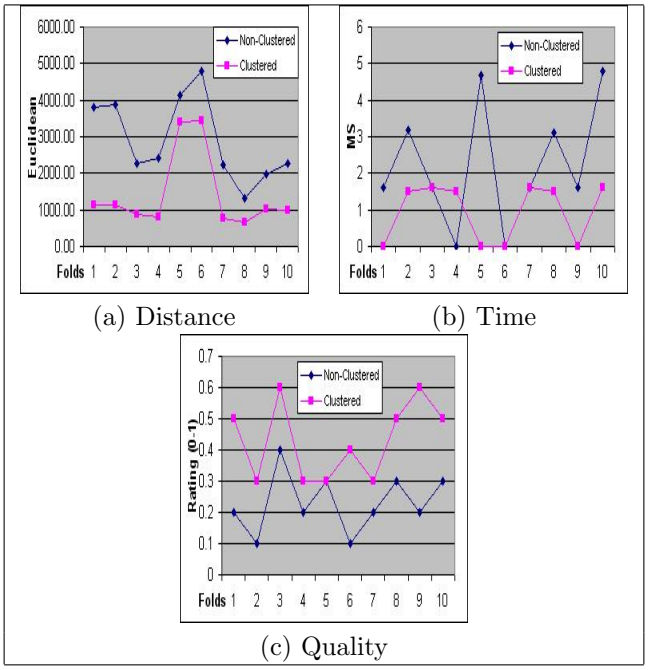
Table 4.12: Video 6: 10 fold Non-Clustered vs. Clustered Non-region Raw Data



(a) Distance (b) Time

(c) Quality

Figure 4.20: Video 6: 10 fold Non-Clustered vs. Clustered Regions



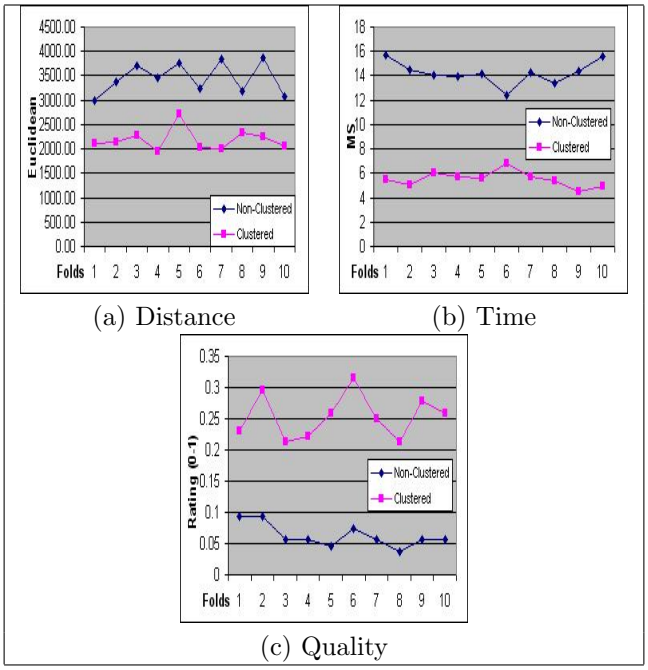
(a) Distance (b) Time

(c) Quality

Figure 4.21: Video 6: 10 fold Non-Clustered vs. Clustered Non-regions

	Non-Clustered			Clustered		
Folds	**Distance**	**Time**	**Quality**	**Distance**	**Time**	**Quality**
1	3000.00	15.69	0.09	2100.00	5.47	0.23
2	3368.00	14.46	0.09	2152.00	5.01	0.30
3	3704.00	14.00	0.06	2268.00	6.03	0.21
4	3451.00	13.91	0.06	1935.00	5.75	0.22
5	3746.00	14.19	0.05	2718.00	5.57	0.26
6	3237.00	12.43	0.07	2031.00	6.76	0.31
7	3829.00	14.29	0.06	2004.00	5.67	0.25
8	3196.00	13.37	0.04	2339.00	5.39	0.21
9	3874.00	14.36	0.06	2262.00	4.54	0.28
10	3065.00	15.58	0.06	2050.00	4.91	0.26
Avg.	**3447.00**	**14.23**	**0.06**	**2185.90**	**5.51**	**0.25**

Table 4.13: All Videos: 10 fold Non-Clustered vs. Clustered Region Raw Data



(a) Distance (b) Time

(c) Quality

Figure 4.22: All Videos: 10 fold Non-Clustered vs. Clustered Regions

non-clustered, thus showing a better degree of quality. Also the retrieval time was less for the clustered set. We also performed the same type of tests between clustering and non-clustering with non-region images. Non-region images refer to the novel images generated by the VENUS system. Figure 4.23 is a graphical representation of Table 4.14.

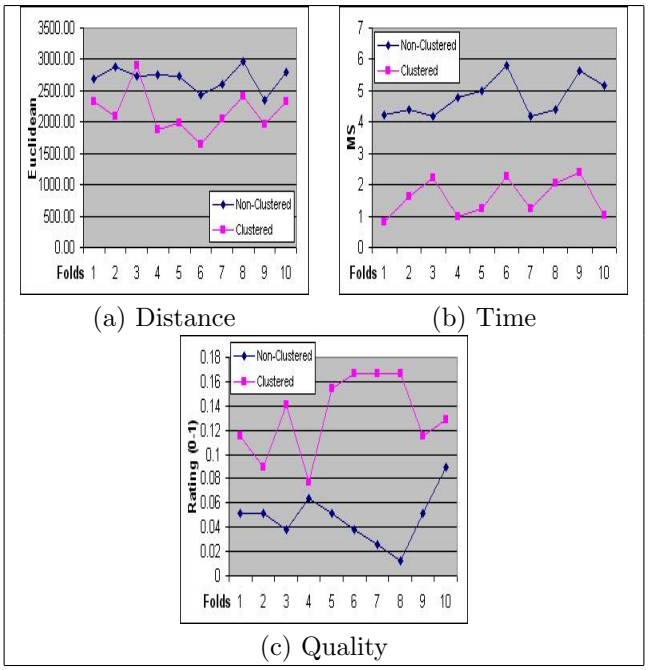
4.3.2 Clustering vs. Non-Clustering

Clustering on connected components have a smaller distance between the query and the retrieval image than non-clustered connected components shown in Figure 4.24. On average clustering also shows less time to retrieve an image and a higher degree of quality in Figure 4.24. The best values for each metric was found and compared together for regions and found similar results (Figure 4.25).

Clustering was also experimented with novel key-frames which produced similar average (Figure 4.26) and best (Figure 4.27) results as the connected component data set. The time retrieved for some videos was zero since the amount of novel key-frames was very low 4.27.

	Non-Clustered			Clustered		
Folds	Distance	Time	Quality	Distance	Time	Quality
1	2684.59	4.21	0.05	2334.05	0.79	0.12
2	2890.68	4.41	0.05	2081.38	1.63	0.09
3	2731.95	4.19	0.04	2912.73	2.23	0.14
4	2751.09	4.79	0.06	1869.58	0.99	0.08
5	2739.05	5.00	0.05	1988.54	1.22	0.15
6	2429.04	5.81	0.04	1632.83	2.24	0.17
7	2612.99	4.17	0.03	2050.76	1.22	0.17
8	2956.68	4.41	0.01	2409.28	2.04	0.17
9	2357.46	5.62	0.05	1958.81	2.38	0.12
10	2792.01	5.17	0.09	2324.92	1.03	0.13
Avg.	**2694.55**	**4.78**	**0.05**	**2156.29**	**1.58**	**0.13**

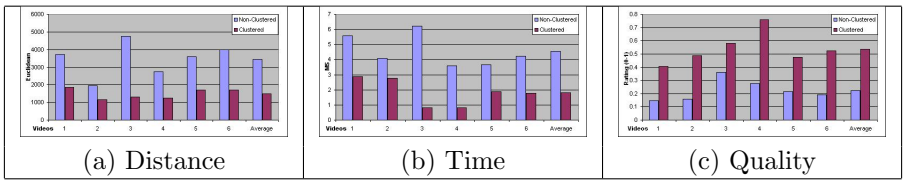
Table 4.14: All Videos: 10 fold Non-Clustered vs. Clustered Non-region Raw Data



(a) Distance (b) Time

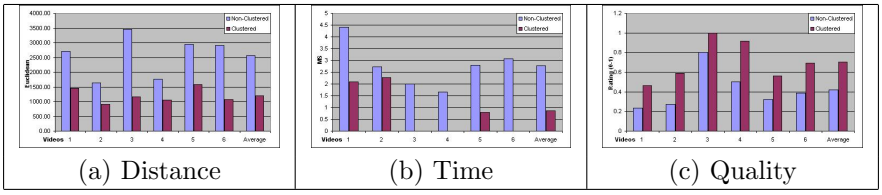
(c) Quality

Figure 4.23: All Videos: 10 fold Non-Clustered vs. Clustered Non-regions



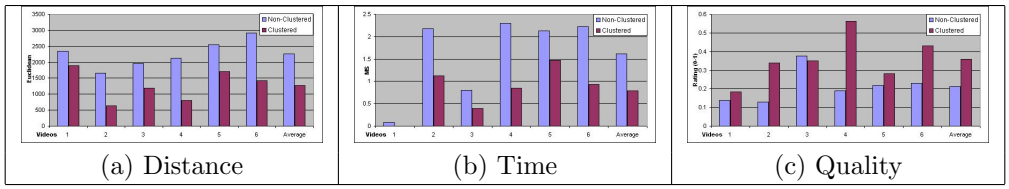
(a) Distance (b) Time (c) Quality

Figure 4.24: Average Clustering vs. Non-Clustering Connected Component Regions



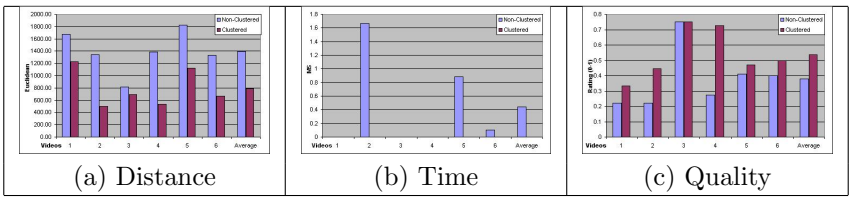
(a) Distance (b) Time (c) Quality

Figure 4.25: Best Clustering vs. Non-Clustering Connected Component Regions



(a) Distance (b) Time (c) Quality

Figure 4.26: Average Clustering vs. Non-Clustering Novel Key-Frames



(a) Distance (b) Time (c) Quality

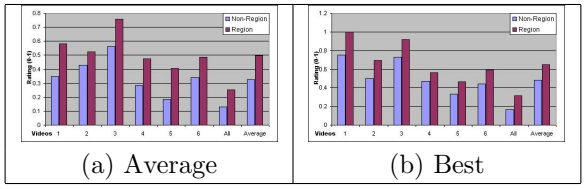
Figure 4.27: Best Clustering vs. Non-Clustering Novel Key-Frames

4.3.3 Connected Component Regions vs. Key-Frame Non-Regions

Experiments were performed to determine the utility of using JPEG 2000 to extract components from the key-frames against key-frames not using JPEG 2000. Quality was used to compare the two data sets since distance and time has no relation since the image size is different and retrieval time is not related in this instance. Our results show that the average and best quality for clustering regions outperforms the retrieval quality for the clustered key-frame non-regions (Figure 4.28).

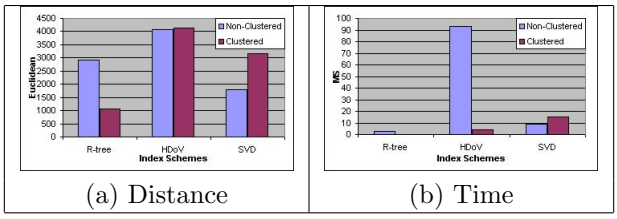
4.3.4 Index Comparisons

Our original approximate indexing using a R-tree was compared to three other of our own implementations of the HDoV-tree, SVD based indexing, and P-tree. (NOTE JMK: P-tree results will be placed here by 5/1/05) Out of all the 10 fold testing we performed on six different video sets, we chose our best case from the R-tree implementation and used this test set for our three other implementations. Our first test we performed used Non-regions which used the metrics Distance and Time shown in Figure 4.29 which showed that each algorithm performed well based on the type of data set given. Our second test was based on a regions data set also using Distance and Time shown in Figure 4.30. Detailed explanations for each test and their results are explained in the summary section for discussion.



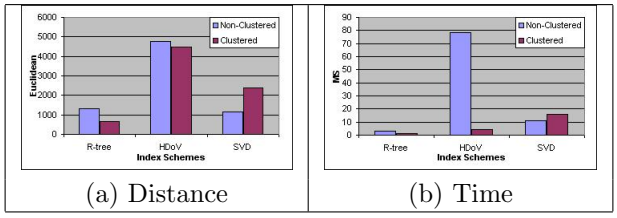
(a) Average (b) Best

Figure 4.28: Clustering Connected Component Regions vs. Key-Frames



(a) Distance (b) Time

Figure 4.29: Index Comparisons for Regions: Clustering vs Non-Clustering



(a) Distance (b) Time

Figure 4.30: Index Comparisons for Non-Regions: Clustering vs Non-Clustering

Based on our experiments, using regions as our data set performed better than non-regions (Figure 3.6). These results confirm our previous experiments between region and non-region data are correct. However, these results show that machine novelty detection systems are very far from matching human perception. The most apparent difference that was observed between the human and machine novelty was in the rate of habituation for novelty detection. Since, the machine-based system uses a habituation technique where new novel events are "remembered" for a while, the significance of novel events wrt the current novelty within the video degrades over time.

Chapter 5

Archiving Novelty Indicies using the JPEG 2000 Standard

JPEG 2000 (JP2) is a new method of compressing images more efficiently than any other image format such as JPEG, GIF, PNG, etc. The main reason this format is in need for investigation is it allows information to be embedded within the image itself. The types of information can essentially be anything electronic such as text, audio, video, images, etc. The following section describes how we incorporated this technology with the use of storage of image information and retrieval to obtain a meaninful representation of a single event. Then we explain the tools we developed to prove that JPEG 2000 is capable of embedding information within the image, extracting, and searching based on image annotations.

5.1 Storage and Retrieval

Currently image annotations are stored and collected side by side. Even though this method is very common, it brings up a lot of risks and flaws. Imagine if medical images were annotated by doctors to describe a tumor within the brain, then suddenly some of the annotations are lost. Without these annotations, the images itself would be useless. By embedding these annotations within the image will guarentee that the description and the image will never be seperated.

In our system, we are embedding the region coordinates, feature sets, and cluster information into the JP2 key-frame image. The annotation information is stored as XML which allows easy extractability of this information to create our index and use these images to extract regions efficiently. These JP2 files is our underlying storage system for all the pre-processing data that has been created for our image query engine.

5.1.1 Retrieval

The query image contains the same features sets as the images in the data set. The shortest Euclidean distance between the query image and the representative image of each cluster is

used to determine which index is to be queried upon. Our results have shown that by using this scheme provides a higher performance than without clustering the data set. Also, by having regions in the data set improves the retrieval quality.

Figure 5.1 shows an example of a query region taken from the original data set and able to retrieve similar images. But in Figure 5.2, a novel key-frame search and it retrieves unrelated images since there are multiple events occuring in a single key-frame. Thus, the features do not describe a singe event and its harder to return a similar match.

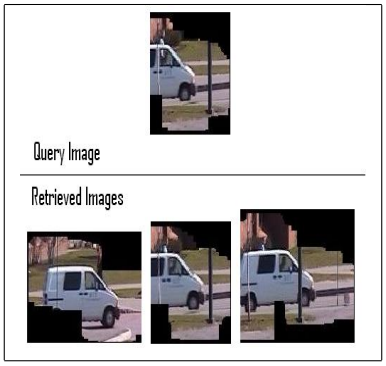


Figure 5.1: Example of a clustered region query

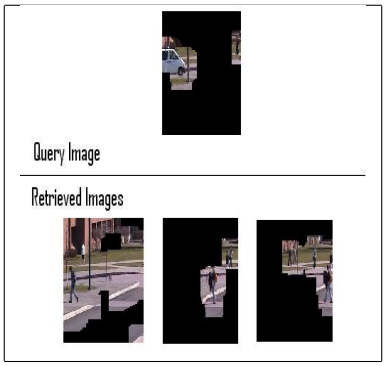


Figure 5.2: Example of a key-frame Query

5.2 Annotation of Region

Initially working with Visual C++ .net is not an easy task. It took awhile to understand the interworkings of Kakadu and the .net package. After investigating this application, pieces started falling together. Figure 5.3 shows a flow chart diagram of the annotation process. Initially the user executes the annotation software an annotates an image. Then the user can upload this image to a server, where at the server a program will then extract the annotations from the image and store it within an Oracle database.

The first major addition to the Kakadu kdu_show application was the addition of the new fields shown in Figure 5.4. In order to add the current date, this had to be modified

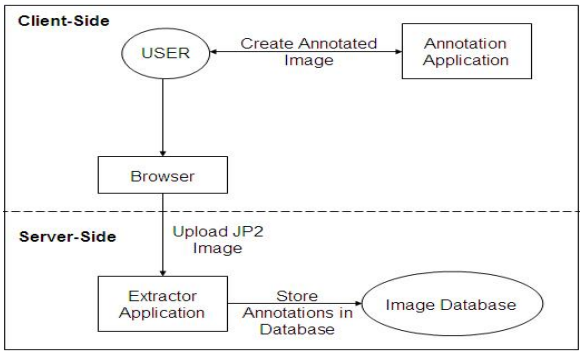


Figure 5.3: This is a flow chart diagram describing the phases of how an annotation of an image is made.

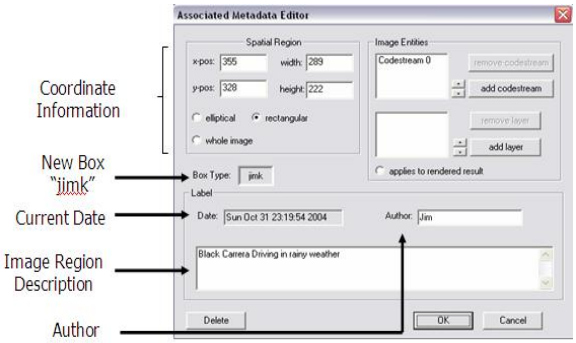


Figure 5.4: This is the gui of when metadata is added to a region.

in several places. First, the gui had to be modified to accomadate the new field. This field cannot be changed. Once a new region is selected, the current system date is populated within the text field. When a region is modified, the date is extracted from the image stored in a XML format. Other text fields were added such as the author and the keywords fields. A region is added to the image when you select a region with the cursor. In the menu bar, select edit then "add metadata". A new form is shown and there you can specify the author's name and the keywords. Once the "OK" button is pressed, the information gathered from the author, keywords, date, and region of location is put in XML format. The region of location is the x and y coordinates for the upper left hand corner. The width and height of the region are the two other coordinates. This data can be viewed by selecting the View then Display metadata from the menu shown in Figure 5.5. There can be multiple regions selected within the .jp2 image. In theory, there can be an infinite amount of regions selected for any region and regions can overlap each other.

Once an image is annotated, these images can be modified. By using kdu_show, the base applications allows you to move your curser over a annotated region and when pressing the ctrl key the keywords are displayed. Since in our implementation, the information is stored as XML, the tags had to be stripped away. When pressing the ¡ctrl¿ key, then clicking

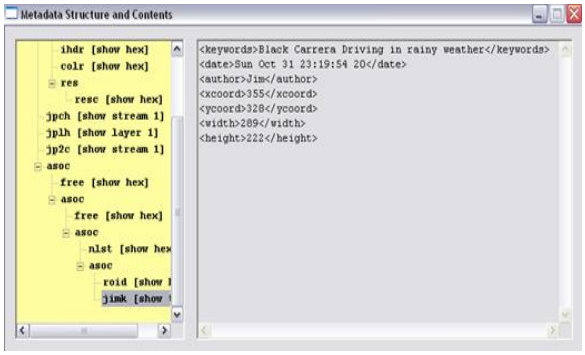


Figure 5.5: This is an example of how the data is stored in XML for each region.

on an annotated region. A new window similar to Figure 5.4 is shown. Each of the data fields (Author, Keywords, Date, and Region of Interest) is taken from the XML tags and displayed within the form. If there is no data within the tags, default data is shown.

Within the metadata view shown in Figure 5.4, a new box was created called "jimk". The box name is limited to four characters. Originally Kakadu used this name as "lbl" for the box. A new name was created to show that a new box can be created and specified as just the metadata for the region.

5.2.1 XML Structure

Currently, the data within each annotation looks very similar to Figure 5.7. In this design a design or a DTD is not setup for this structure of the XML. If the name of the XML tag needs to be changed, there are several places that needs to be modified. First of all, the method OnOK for the metadata editor class needs to be modified. Within this method it describes the layout of the XML Structure. Once the name is changed, the XML Parser that extracts the information needs to also be changed. This is to ensure that the correct information is pulled from the image.

Since changing the name of tag is quite convoluted, a future enhancement needs to be done to improve the ease of change XML tag information. Either a standard DTD needs to be created where both the annotation and XML Parser programs can reference for the name of the tag.

5.2.2 Kdu_show.cpp

There were two main methods that were changed, the methods are references in Figure 5.6. The show methods provides functionality when you press the ¡ctrl¿ key over a annotated region, the keywords specified for the region will show up. An example is shown in Figure 5.8. Since the new application stores everything in XML format, the correct data has to be pulled out and displayed to the screen. The keyword information in this case is parsed from

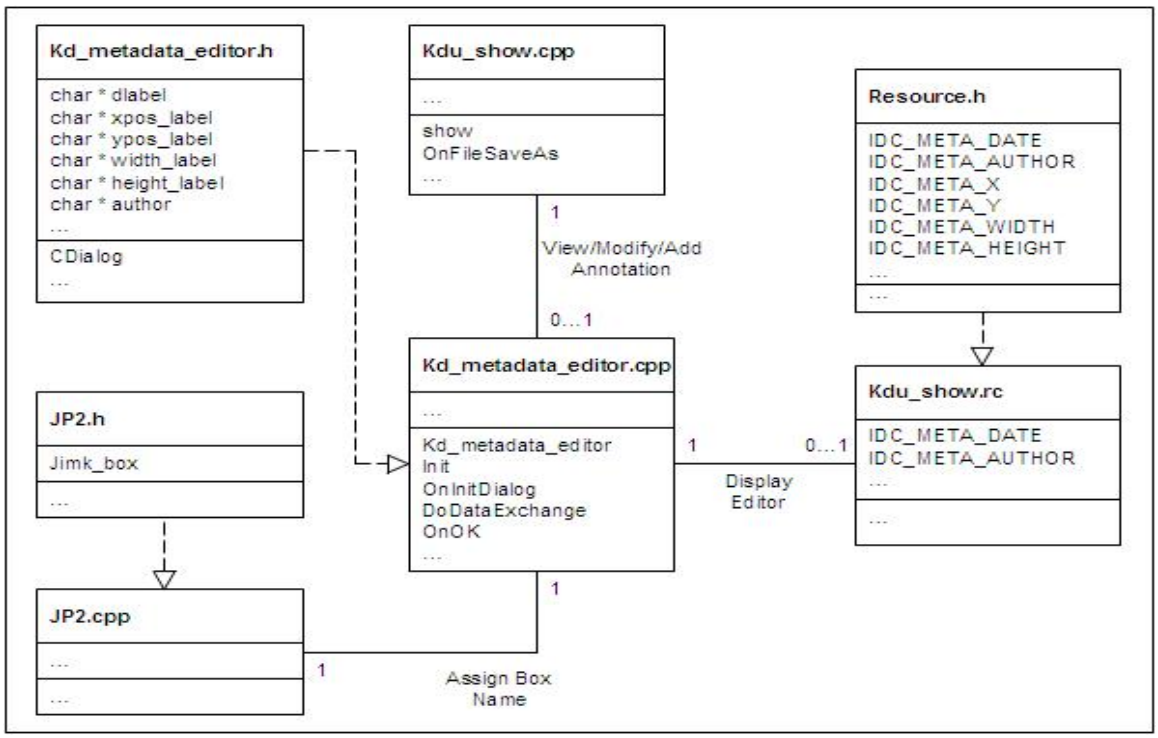


Figure 5.6: Modified Class Diagram. Out of simplicity sake, this diagram consists all the changes made within the Kakadu system. It doesn't show all of the classes within Kakadu since there are a couple of hundred classes within this application and the class diagram would be very confusing. The diagram starts at Kdu_show and when a region is selected, the kd_meta_data.cpp is shown. Also, to view/modify a region, the kd_meta_data.cpp is also shown. Then the mfc form kdu_show.rc is displayed for the user to add/modify/view the annotation. JP2.cpp simply shows that a new box was created for "jimk".

the XML information.

The second main change was within the OnFileSaveAs method. The main problem Kakadu has is that it doesn't allow you to save an annotated image in a .jp2 format. It will only allow you to save it as .jpx. This causes a problem for the search phase of the project. The search phase requires a browser to view .jpx files. There are a few plugins to view this image, but .jp2 is far more popular. A workaround to this problem was instead of saving the file as .jpx, the file was saved as .jp2. The file itself is still jpx, but the extention will say .jp2. This is still a future enhancement to save the file with the annotations in a jp2 format.

5.2.3 Kd_metadata_editor.cpp

This class had the most significant changes among all the other classes. When a user has selected a region from an image (Figure 5.8), and clicked on the region. The metadata editor form is displayed to the user (Figure 5.4). This allows the user to either add/modify/view the data of that region.

<keywords>Black Carrera Driving in rainy weather</keywords>
<date>Sun Oct 31 23:19:54 20</date>
<author>Jim</author>
<xcoord>355</xcoord>
<ycoord>328</ycoord>
<width>289</width>
<height>222</height>

Figure 5.7: This is an example of the XML annotation information stored within the image.

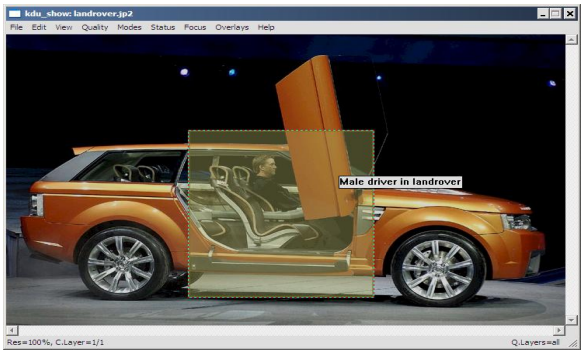


Figure 5.8: This is an example of when the ¡ctrl¿ key is pressed over an annotated region.

The kd_metadata_editor constructor method had to be modified to populate default values within the form fields. Originally, Kakadu had only one field to enter data from the user. In this application, there's two new fields that were added which are the author and keywords. It was found that Kakadu has a bug in their system on where it will not accept any blank fields. Thus, default values were added to be sure each field was filled in.

The second method that was modified was the Init function. This function's role is to look into the image and determine if there are any exisiting information about this region. If there are, it finds the information and brings it to the class level. Since there are two fields now, the code was modified to access both field information.

The OnInitDialog is responsible for populating the data within the form if data was found in the Init function. Since the data is now in XML format, the data had to be parsed out and then displayed into its respected text box.

DoDataExchange is a simple method where it coordinates the data that is requested from the Init function to the image itself. Since there are additional fields that were added, the code was modified to handle multiple types of data between the image and this class.

Method OnOK was the most significant change within this class. OnOK validates the data within the metadata editor form and then sent back to kdu_show. Since there were a couple of fields added and the data needs to be in XML format, there were numerous

changes within the code to handle this. Initially the data (X, Y, Width, Height, Author, Keywords) from the form and translated into an XML format. The data is then stored within its respected labels and sent back to kdu_show.

5.2.4 Kdu_show.rc

This class dealt with the gui side of things for this application. The only gui form that needed to be modified was the metadata editor form. An additional field was added for the author which is meant for the person that added a specific region within the image. Also, the keyword textbox was modifed to allow multiple lines to be added for the annotated region. The system date is also displayed to the user if it is a new annotation. If its an existing annotation, the date when that was created will be showed. This field is shown only for display and cannot be edited. Resource.h had to be modified to allow additional fields within the form. Kakadu already had the fields of X, Y, Width, and Height, but they were not doing anything with those fields. Additional code was added to maintain that information and store it within the image. The (X,Y) represents the top left hand corner of the region. Width and height measures the size of the whole region.

5.2.5 Jp2.h

In this application, a new box was created to hold all of the annotation information. Within this file holds all the different definitions of each box. In order to create a new box, a box name of "jimk" was added to the code. Currently in Kakadu, the limit of the box name is four characters. The name of the box can be easily changed, but cannot conflict with other box names.

5.3 Upload and Extraction of Region

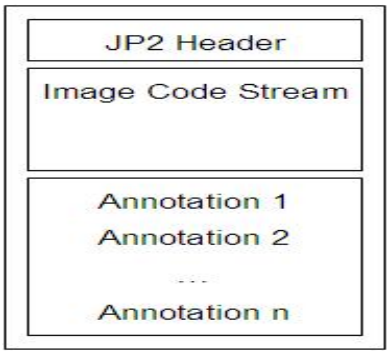


Figure 5.9: This is the general file structure of a JP2 image. The first initial boxes contain the header information such as file type, resolution, and image size. The next box contains the codes tream of the image. At the end of the file contains the annotation information. There can be an infinite amount of boxes within this image.

Once an image was created with annotations, a tool needed to be developed to extract this information from the image. A java program was created to find these "jimk" boxes within the image. According to the JP2 image format (Figure 5.9), the boxes within the image are always setup the same way. The very first four characters in the image represent the box size for the name of the next four characters. This pattern continues from all the header boxes, the codestream box, and finally the metadata(jimk) box. The java program skips bytes through the file until it reaches a "jimk" box. This is done to be sure that scalability of larger images does not have an issue with the extraction program. Once a "jimk" box is found, the XML data is retrieved. The tags and information is then parsed out and ready to submit to a database. The information extracted from an image is then stored into an Oracle database.

In order to automate this process, the java program was linked to a server side scripting language (php) to upload any annotated JP2 image. Once the image is uploaded and stored on the server, the java program is executed and extracts all the annotated information. This information is then sent to the database.

5.4 Search Images

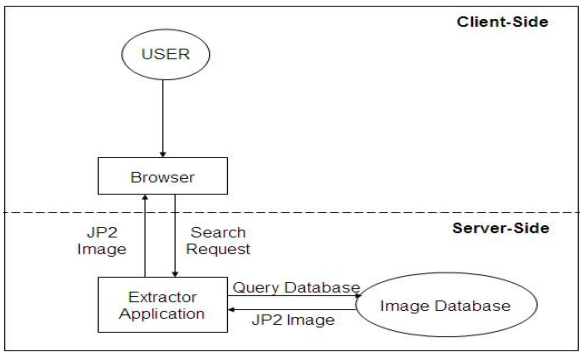


Figure 5.10: This is a flow chart diagram of how the search phase works.

A search utility was created in PHP to search for the annotated information (Figure 5.10). In essence, this search utility program will give better results than any other image search engine on the internet, even google. Since regions were annotated by experts, and potentially more than one annotation can describe an image. Currently the whole image is returned to the client. Within the server, it can specify the number of bytes to be sent over to the client. The client can then ask to improve their image and the server will send more bytes of data to the client. This is a very useful feature when the images are quite large (ex. in gigabytes). Thus even a dial up connection could see the full image at low resolution.

Chapter 6

System Implementation

The Query Engine of Novelty in Video Streams has been a year long project. This section will describe each step in terms of the implementation and its execution. The implementation of this project uses the programming languages of Visual C++ .net, Java, Matlab, and PHP. There are six main sections within this chapter to produce the correct data set and its execution, three steps for preprocessing, then the 10 fold data sets and query engine execution.

6.1 Region and Feature Extraction

This is the initial step of the preprocessing that needs to be done before any tests can be conducted. In figure 6.1, this step is shown in detail from the novel images generated from a video stream to regions and features being extracted. This section will go into detail of every step in this figure.

6.1.1 VENUS' Novel Frames

The VENUS system takes as input a uncompressed AVI video which produces novel frames such as in Figure 1.1 in Chapter 1 of this thesis. If the AVI file is not uncompressed, I have used a 3rd party software called VirtualDub [2]. The compressed AVI file can be opened using this program and saved as an uncompressed AVI file. Before this system can be executed, a couple methods need to be modified. Within the main.m file of VENUS, the directory to where the videos must be changed. Also, the directory to the mat files need to be defined. These mat files are meant as a temporary directory VENUS needs to store its frames as its executing.

Also, I've made a modification to this program so it will generate the novel frames as the system is processing the video. The file I have created is called saveToBMPs.m, this program is called within the main.m program. The path to save each individual novel frame must be specified in this program. For every frame that causes the size to be greater than

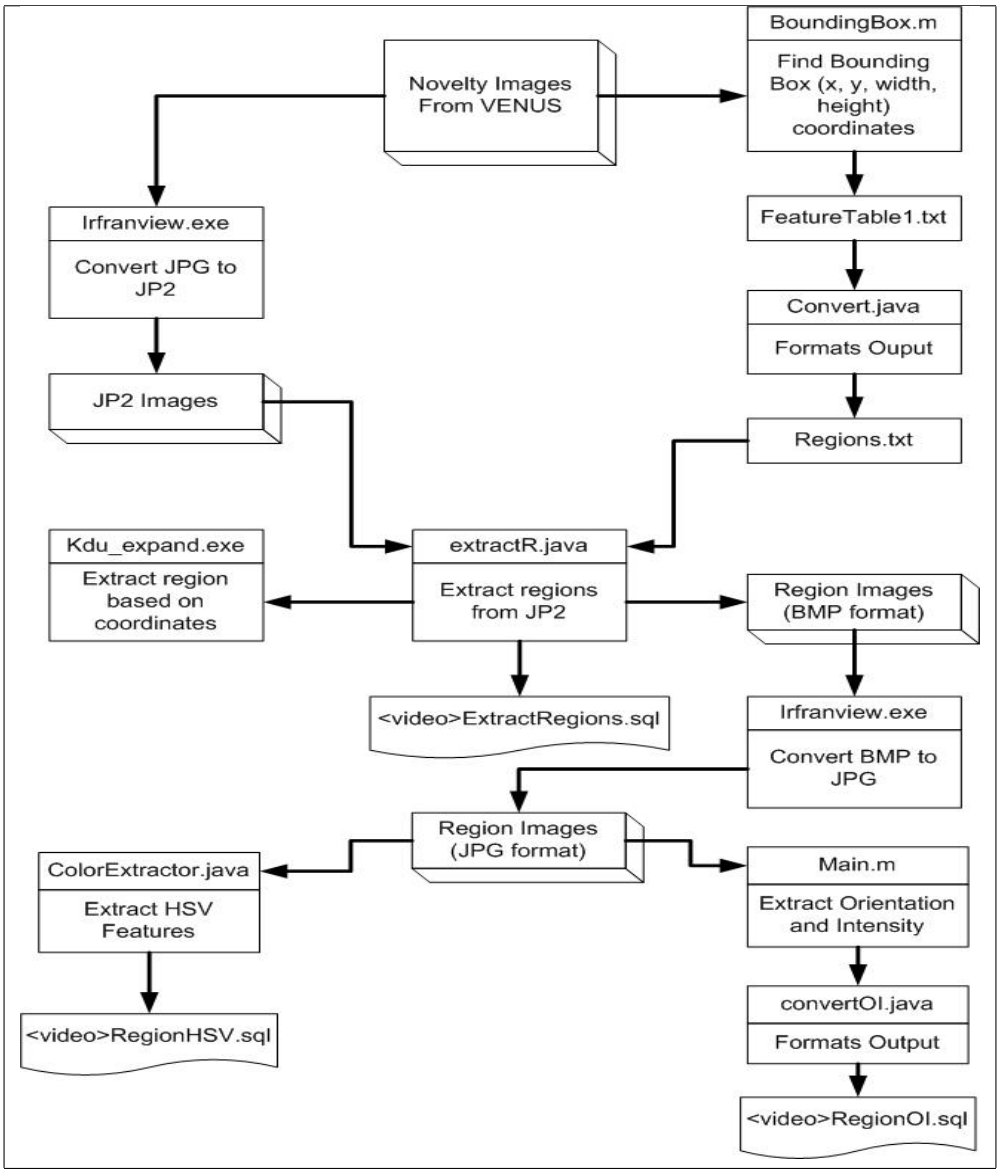


Figure 6.1: Step 1/6: Region and Feature Extraction

one, it will generate a novel frame to the specified directory. Within phase 1 of Figure 1.1, notice the frame that is all black. This will have the size of zero, where all the other frames that has one or more visual components will have a size greater than zero. Currently the novel images are saved as JPEG format, but this can easily be changed by modifying the extension within this program.

6.1.2 Connected Components

Once the novel images are created, I have created a Matlab program that will extract the regions or connected components from the image. Connected components are regions within a novel frame that are together or near each other. The goal of this step is to extract these regions as separate images. This will allow the features for the extracted image describe a specific event instead of a couple of events.

This program name is called BoundingBox.m and is also specified in Figure 6.1. The path of where the JPEG files are stored (same path as saveToBMPs) must be specified. This is a stand alone program, unlike VENUS and thus can be executed anywhere.

Initially each image is read into the program and stored into an array. Each image in converted into a binary format where the zero values are the black regions and ones for everything else. There is a threshold to determine this distinction. For each non-zero group of values, a square region is founded. This program uses the concept of 'open' where if the sides of the square regions touch another non-zero group, the square will get larger and combine the other group.

Once the square regions are created, each region is labeled in ascending order. Then statistical information is gathered by each region such as the (x,y) coordinated, width, and height. The (x,y) coordinates signify the top left hand corner of the region. Each labeled region is then writted to a FeaturesTable2.txt file. Convert.java will format this output to a cleaner input file that can easily be read for the next step. Convert.java is executed by this format: $java\ Convert\ <input\ filename>\ <video\ name>$

6.1.3 Region Extraction

Before regions can be extracted from the novel images. The original novel images must be converted into JPEG 2000. Since the fastest method of extracting regions is by using JPEG 2000. It is possible to use other image formats such as BMP or TIF, but these images need to be uncompressed first. This process may take some time to extract. This simple type of conversion can be done by a 3rd party software called Irfanview [36]. This program allows for a batch conversion to convert all the novel JPG images to JP2 very quickly.

The actual extraction is executed by using the system called Kakadu [39]. The program executable I am using from this system is called Kdu_expand.exe. Based on a set of (x,y)

coordinates, width, height and the original image, it can extract a region from a JPEG 2000 file to a BMP file format.

The extractR.java program will read the Regions.txt file created by the BoundingBox.m program, find the correct image from the converted JP2 novel images, and execute the Kdu_expand.exe program. However, the coordinates generated from the BoundingBox.m program are not the exact values used to call this program. The JPEG 2000 file contains header information to extract the actual width and height of the image. JPEG 2000's information is broken into different boxes, such as the header box or image stream box (For more information, please refer to the JPEG 2000 chapter of this thesis). The box that contains the header information is called 'jp2h.' The usage for this program is: $java\ extractR\ <regions.txt>\ <video\ name>$

The arguments required for the Kdu_expand.exe are percentage values of each coordinate based on the actual size of the image. The arguments are x, y, width, and height. The x value is calculated by the region x value divided by the original image width. The y value is found by the region y coordinated divided by the original image height. Width is found by dividing with the original width and height is found by dividing with the original height. The Kdu_expand.exe is executed by:
$Kdu\_expand -region\ \{x,y\},\{height,width\} -i\ <\ input\ JP2\ file\ >\ -o\ <\ output\ BMP\ file\ >$
.

For each image that is created, a sql command is created. The ¡videoExtractRegions.sql will contain each region id, the coordinate information (the calculated values), and video name. This is one of the three files needed for the clustering step. The next step of this phase is the feature extraction.

6.1.4 Feature Extraction

There are two main programs that handle the feature extraction for an image. The first program is called color extractor which extracts Hue, Saturation, and Value (HSV) features. Hue has features of what we perceive as color. Saturation is based on the amount of grey, white, black that is combined with the color. Value is the amount of brightness found within the image. This program is executed by: $java\ ColorExtractor\ <folder\ name>\ <video\ name>$ The folder name consists of the novel images that are converted into a JP2 format. The video name is needed since this program will create the RegionHSV.sql file. This file will contain the region id and its relating HSV features.

A matlab program is used to create the Orientation and Intensity features of the novel image. The directory used for the previous program must be specified within the Main.m program of the Create Features package. The orientation features are generated by using the Gabor orientation filters to extract the edges of 0, 45, 90, and 135 degree orientations. An

intensity contrast filter is used to gather these features. The program outputs also outputs RGB features, these features are ignored in this system. Once the features are created, they are formatted and converted using the convertOI.java program. This program will create the RegionOI.sql file which consists of the region id and its orientaion and intensity features.

6.2 Clustering

The second stage of preprocessing is the clustering step. The clustering was performed using the Oracle database. Oracle was used because the organization that is needed can be utilized easily. Before the clustering can be performed, a set of tables need to be added into the database. Then the sql files generated in the previous step are added to this system, then clustering can be performed. This framework is the general method of clustering the images, the testing section will show how the test data sets were generated based on this framework.

6.2.1 Add Tables

The set of tables added to the database is shown in Figure 6.2. The tables RegionFeaturesHSV, RegionFeaturesOI, and RegionCoordinates contain the data generated for each video in the previous step. Their linked together by the video name. RegionTotalFeatures is used for the 10 fold data set testing. The testing section of this chapter will go in more detail of the use of this table. RegionPreprocessing contains the binary values generated by the CMC algorithm. RegionClusterSumm contains the cluster information. Each region for a video is associated to a assigned cluster number by the CMC algorithm. The RegionCentroid table consist of the regions that are the representative images, generated by the RIA. The centroid value is the cluster number the centroid region is associated to.

6.2.2 Clustering

Once the three sql files generated in the previous step has been added into the database, the clustering can begin shown in Figure 6.3. The video to be clustered needs to be specified within each php program. Clustering.php is based on the CMC algorithm defined in Chapter 1 of this thesis. Initially the features are collected for a certain video from the RegionFeaturesHSV and RegionFeaturesOI. Then each feature's average is calculated based on the gathered feature. For each image, the feature is compared to against the threshold. If the value is greater, it is assigned a one, and otherwize a zero. These values are added to the RegionPreprocessing table. In this table, it will contain binary value feature sets which are then grouped together. For each group, a cluster number is assigned and added to the RegionClusterSumm table.

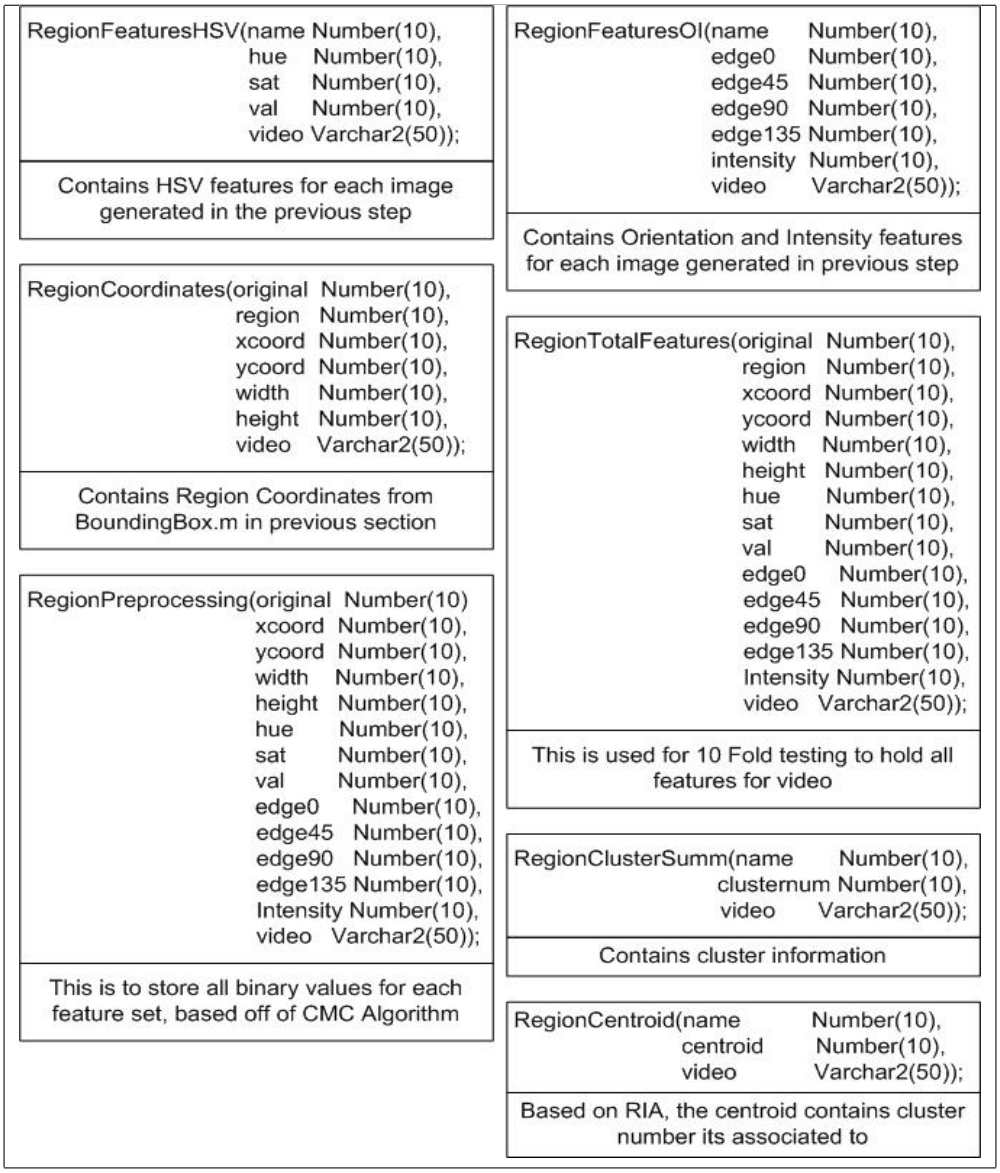


Figure 6.2: Preprocessing Table Structure

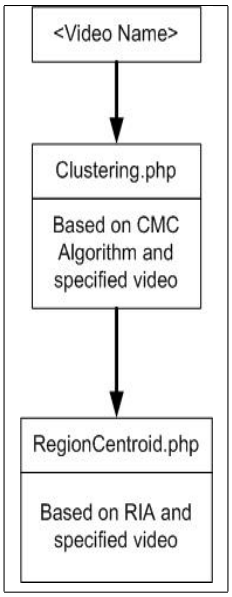


Figure 6.3: Step 2/6: Clustering

The next step is to obtain the best representative image which is based on the RIA from Chapter 1 of this thesis. For each cluster within RegionClusterSumm, the average value of the cluster is found. This point is considered the centroid of the cluster. Then the shortest distance between the centroid and image is found. This image will be the best representative image for this cluster. Then this image is added into the RegionCentroid table along with the cluster number. The clustering is completed and can then continue to the next step of JP2 annotation.

6.3 Manual JP2 Annotation

The base of the application is based of the Kakadu application from the University of New South Wales at Sydney [39]. This application provides the basic framework needed to develop our applications. The main purpose of this application is that the image compression code is already developed and there's no need to reproduce this code since this part is very convoluted. However, there were numerous of problems and compatibility issues that needed to be addressed. The following sections is an intensive explanation of the changes and their purpose to complete each goal addressed above.

6.3.1 Background

Kakadu is an application provided by the University of South Wales at Sydney. Its an open source application which allows us to change and modify to complete our goals. There are numerous applications already built within Kakadu to show its capabilities. The base of our application is based on Kakadu's application called kdu_show. Its an application to

view a .jp2 image file and add metadata content to a specific region of the image file. The enhancements needed to be done in this application to complete our goal is as follows:

- The ability to add more metadata within the image such as the user's name that made the annotation, the date of the change, and keywords describing the annotation

- Instead of having a "lbl" box, create our own box called "jimk".

- Within the "View Metadata", the data stored within the image needs to be in XML format to allow the ease of extracting the information.

- Once an image has been annotated, the image needs to be saved as a .jp2 instead of .jpx.

These changes will be explained in detail within the "Annotation the Region" subsection. Another application within Kakadu that was helpful was the kdu_compress application. This program compresses a .bmp image file into a .jp2 file. There are some limitations to this program such as the input image files, but it does show an example of what Kakadu can do. There are other applications that serve the same purpose called Irfanview [36] which compresses any image file into a .jp2 image. Kdu_compress was not needed to modify to meet our goals above.

6.3.2 Installation

The base application came with installation details explaining how to install Kakadu and begin working on it. There were some issues with the installation which took quite some time in figuring everything out. It failed to mention that you needed a libtiff_i.lib file to be installed within your system before compiling the Kakadu application. This file is freeware and can be downloaded from the internet. This file should be in the root directory of where your Kakadu application is stored. Once installed, simple compile the corsys directory to build on the base classes. Then compile the apps folder which will create all the applications for Kakadu. The appliactions will be stored within the bin folder of the Kakadu directory.

This system is developed mainly using Visual C++. In this application Visual C++ .net was used. The Kakadu application also provides you nativ Java programs which allow you to do pretty much everything with Java instead of C++. The instructions of doing this is fairly simple and explained throughly within the Kakadu installation guide. However, once the code is converted into Java, most of the applications provided in C++ needs to be re-written. Thus, starting from C++ is far more easier and faster to achieve our goals.

6.4 Automatic JP2 Annotation

This is the last phase of the preprocessing steps. This phase will take the information generated from the past two steps and annotated the original VENUS' JP2 image in XML format shown in Figure 6.4. The reason why the original files are being annotated is since

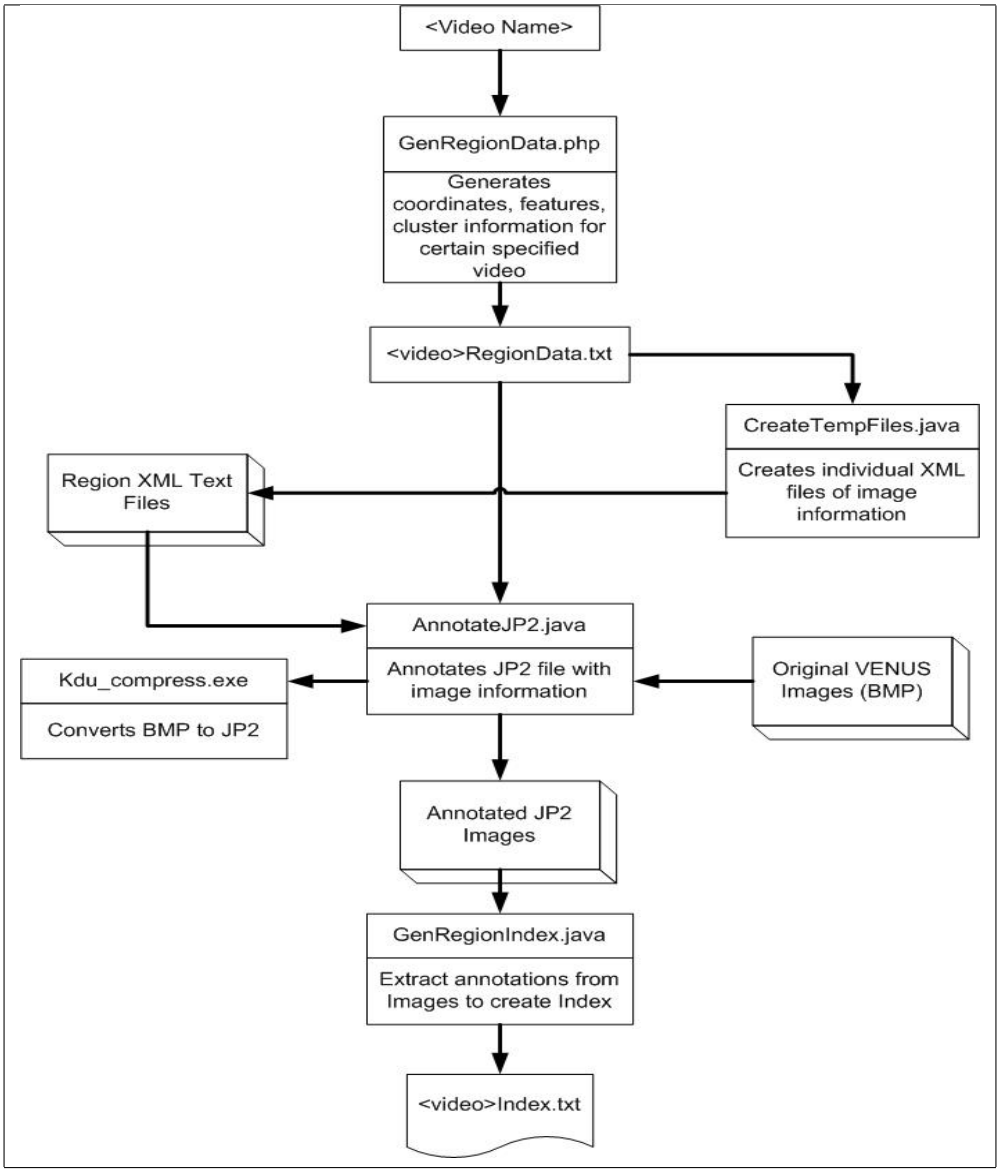


Figure 6.4: Step 3/6: JP2 Annotation

all the information is located for an image is in a single file. If the already extracted regions or the information in the database is lost, the information can be re-generated from the JPEG 2000 files.

6.4.1 XML Files Creation

Since all the information has been generated within the database during the past two preprocessing steps, we can use this information to generated the annotations. Within the program, GenRegionData.php, it will create all the image information such as: image id, region id, x and y coordinates, width, height, HSV, Orientation, Intensity, Cluster Number, Centroid(Note: a value of 1 if image represents representative image, a value of 0 otherwize), and video name. This information is then formatted and placed within the RegionData.txt file.

An individual XML text file is created for each image. This is needed for the next step of annotation. An example of an XML text file is below:

jimk
<original>9</original>
<region>406</region>
<xcoord>.00195</xcoord>
<ycoord>.00195</ycoord>
<width>.17969</width>
<height>.12891</height>
<red>1104</red>
<green>1123</green>
<blue>1095</blue>
<edge0>1641</edge0>
<edge45>1949</edge45>
<edge90>3314</edge90>
<edge145>1700</edge145>
<intensity>4926</intensity>
<clusternumm>30</clusternumm>
<centroid>0</centroid>
<videoname>badday</videoname>

The first line of the text file contains the name of the box. In this system, the name for the xml box is called 'jimk'. The name of the box is to notify the annotation extraction program on where the annotations are stored within the image. The information is stored as xml to allow easy extractibility of the information.

6.4.2 Annotation

Once the XML text files are created, we can then annotate each image with the given XML file. AnnotateJP2 takes three inputs, original VENUS novel images in BMP format, RegionData.txt, and the Region XML Text files. For each image, it will annotate the image using Kdu_compress.exe. It is quite possible to have more than one XML file for a single image. The usage to call Kdu_compress.exe is:
kdu_compress.exe -i <original bmp image>.bmp -o <name of annotated image>.jp2 -jp2_box <XML text file>(,<other XML text files>)
The usage to call AnnotateJP2 is:

java AnnotateJP2 <Regions text file> <video name> This program will then store each annotated image within the directory <video name>_annotate_jp2.

6.4.3 Index Creation

The GenRegionIndex.java program was created to show that the image information can be recreated if the data within the database is lost. Also, an index can be created by using this program. Initially each image is read at a time. For each image, the program will go through the header box, then our 'jimk' box we created, and finally the image stream. Four bytes before the 'jimk' box is the size of the box. Based on the size, the program knows how much to read from the image. The information is then extracted from the XML tags and added to the Index.txt file. There can be more than one 'jimk' box which also goes through this same process. New regions can also be extracted based on the coordinate information.

6.5 Query Engine

Figure shows the general framework of this query engine. This query engine will take as input an image and find the most similar image using an R-tree index to query the information. This framework was created for the use of proof of concept. Then this framework was heavily modified for testing purposes which is discussed in the next section.

6.5.1 Query Parser

The query parser handles on obtaining the image information and sending it to the R-tree class. This program takes in three arguments which are the index files, query table, and the query image. The index files are either generated by the previous step or from the database. The query table consist of images containing features already extracted. For the sake of simplicity, features were already extracted before the execution of this program. Since implementation has been created for feature extraction, it can easily be added to this program to allow any image to query upon.

6.5.2 R-Tree

Multiple indicies are built for each feature within the feature set. A more efficient implementation has been built for an approximate index described in the next section. Based on the query image features, the system will find the nearest element for each index. Since there are eight indicies, it is quite possible to have more than one image retrieved. To find the closent image to the query image, the Euclidean distance was then found between the retrieved image. The shortest distance between the query and retrieval image will be image shown to the user. The system will display the retrieved image, the time for retrieval, and

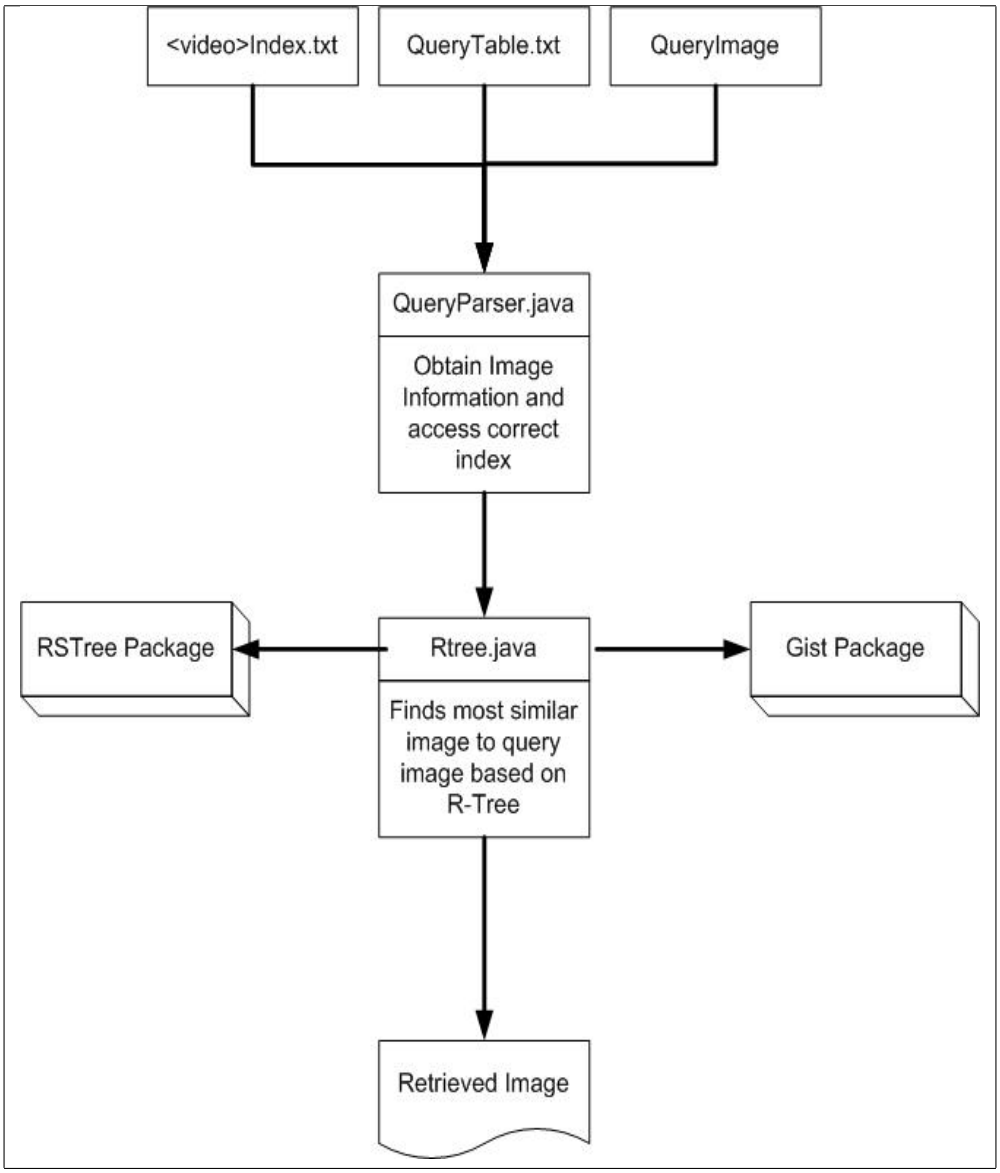


Figure 6.5: Step 4/6: Query Engine General Framework

the Euclidean distance. For more details on how this system is built, please refer to Chapter 4 on R-trees.

6.6 Test Data Sets

The testing of this query engine was based on 10 fold testing using both region and non-region data sets, clustering and non-clustering implementations, and using 6 different video streams. The first part of the testing was the creation of the data sets shown in Figure 6.6. The second part dealt with the modification of the query engine to handle clustered and

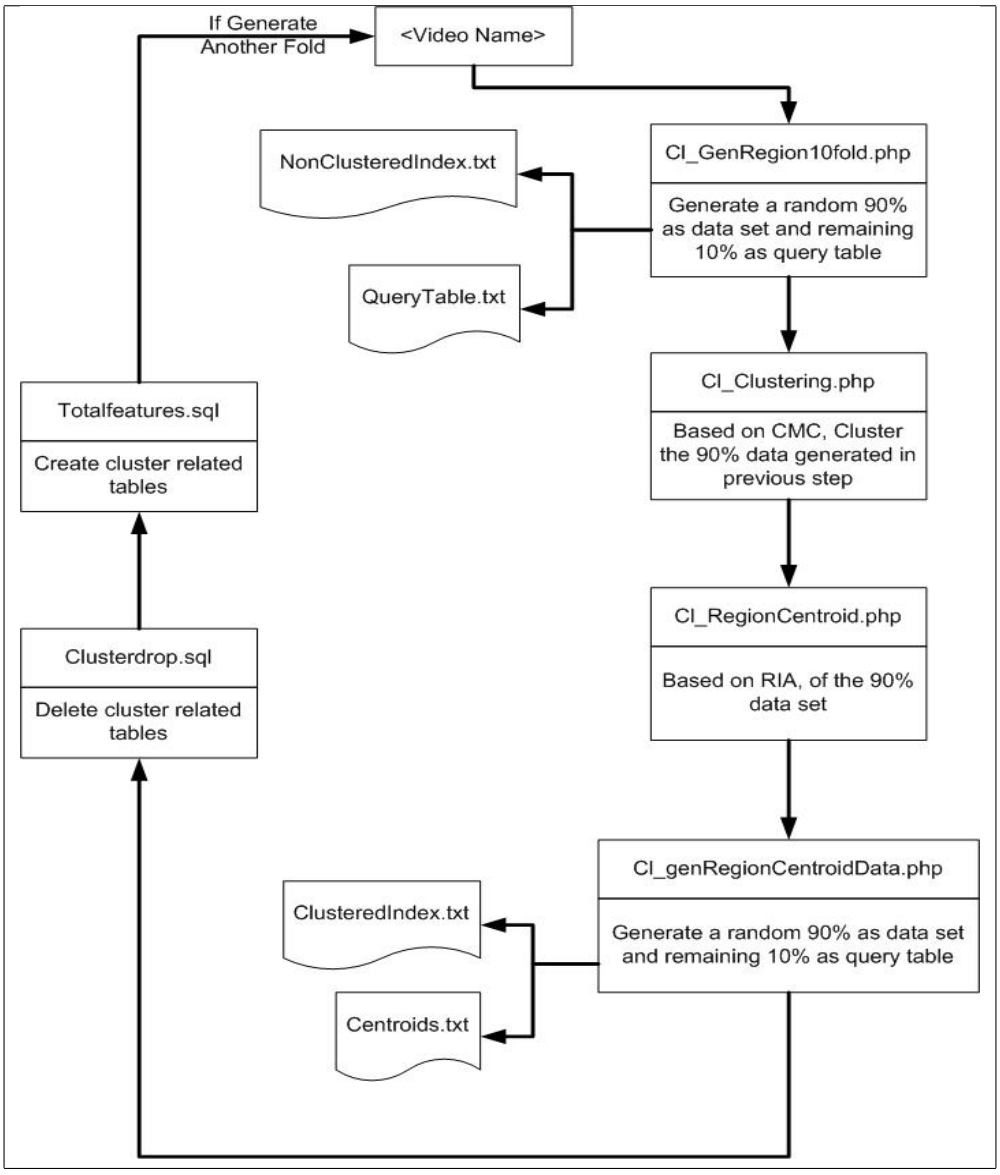


Figure 6.6: Step 5/6: Creation of 10 Fold Data sets

non-clustered data sets. This section will go into detail of how the data sets were created

and evaluated using the modified R-tree implementation.

6.6.1 Non-Clustered Data Sets

Initially for a given video name, Cl_GenRegion10fold.php will randomly choose 90 percent of the data set from the data base. The remaining 10 percent is used for the query table. If the data within the database became corrupted, the annotated JP2 files can re-load the database with the image information. This php file will create two sets of files. The NonClusteredIndex.txt and the QueryTable.txt files. The NonClusteredIndex.txt will contain all the image information except the clustering details. The QueryTable will be used for both the Non-Clustered and Clustered data sets to be sure the results are consistant.

6.6.2 Clustered Data Sets

The 90 percent data set generated in the previous step will automatically place the information within the TotalFeatures table. The Cl_Clustering.php and Cl_RegionCentroid.php will work from this table to find the clusters and centroids. This is a little different from step 2 of preprocessing, since we're working with a random 90 percent of the data set.

Once the information is clustered, Cl_genRegionCentroidData.php will generate the data set used for the clustered data set. This program will generate two files called ClusteredIndex.txt and Centroids.txt. The ClusteredIndex.txt will contain the features information and the cluster number the region is associated to. The Centroid.txt file will contain the best representative images for each cluster, along with their feature set.

6.6.3 Fold Tests

In order to create another fold with the same data sets. The information within the TotalFeatures, ClusterSumm, and RegionCentroid need to be deleted and recreated. By simply reloading Cl_GenRegion10fold and going through Figure 6.6 again will recreate another test set. For this testing, I have executed this ten times for each video stream.

6.7 Testing

The Query Engine implementation was modified to quickly retrieve the results based on the given data sets. Figure 6.7 shows the flow chart of the modified query engine. The modification allowed to test for both clustered and non-clustered data. It also allowed to measure the quality fairly quickly and self generate an excel file with all the statistical information.

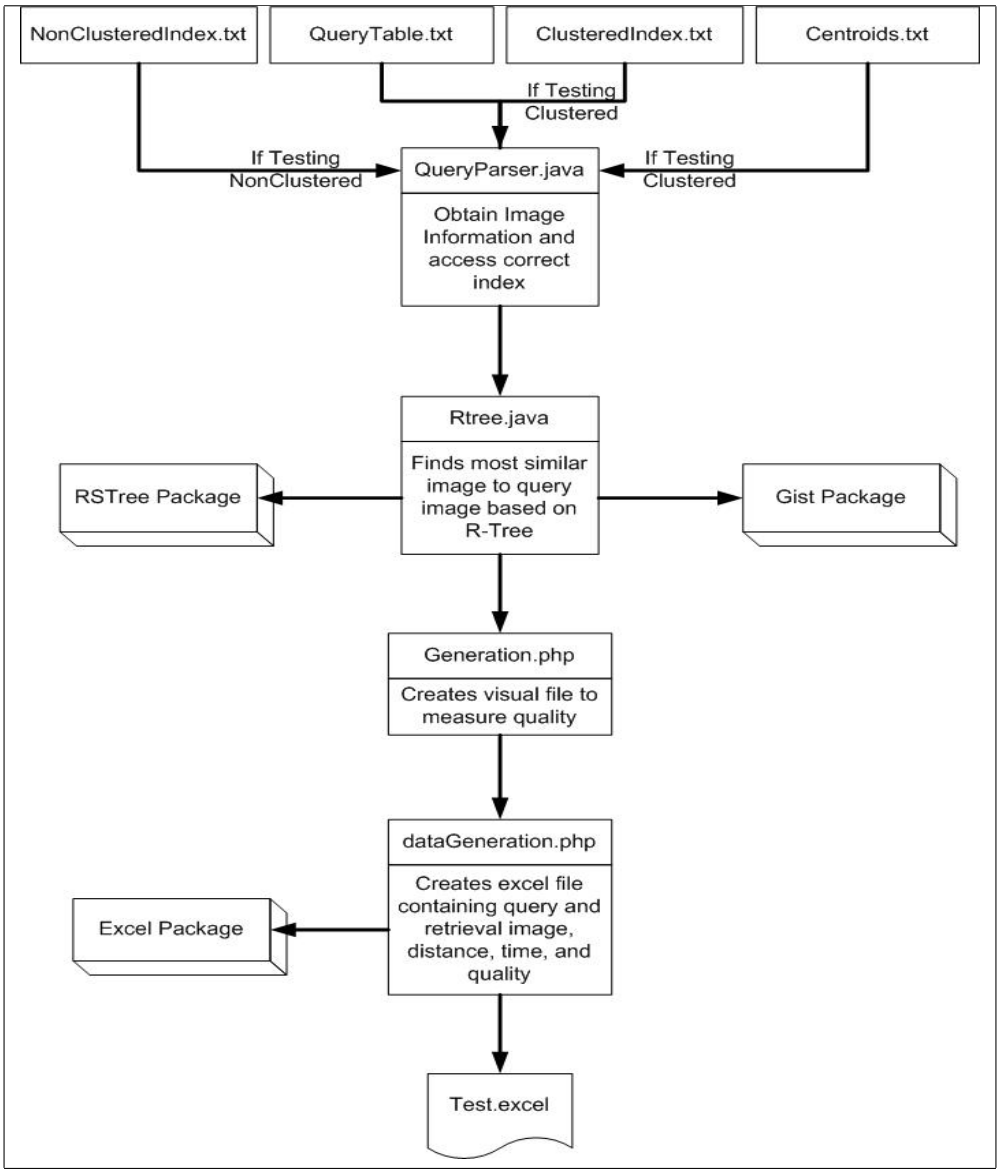


Figure 6.7: Step 6/6: Query Engine Testing

6.7.1 Approximate Clustered Index

This type of an index requires three different input files which were created and described in the previous section. The three files are the QueryTable.txt, ClusteredIndex.txt, and Centroids.txt. This process is called Approximate, since the index is based on a single feature. This is different from the original query engine design. Since the information is already clustered, only one attribute is needed for the index. However, a single index is created for each cluster in the data set. This will improve quality and retreval time which is shown in the results section of Chapter 1. The query table consist the 10 percent remaining data set which will be queried within this approximate clustered index.

For each image in the query table, it will find the shortest Euclidean distance between the query image and each of the centroids found in Centroids.txt. Then it will query on a single index within the RTree.java class for the most similar image.

6.7.2 Approximate Non-Clustered Index

In order to obtain a comparable result for a non-clustered data set, an approximation was also used. Since the data is not clustered and a single index will hold all the images, the quality and retrieval time will be worst. This is proven by the results section in Chapter 1.

The index is built on a single attribute and queried upon by the query image. The R-Tree design is the same for both Approximate and All indices.

6.7.3 Excel Generation

Since there was an abundant amount of data sets and the degree of quality was done manually. The process of generating the results had to be as efficient as possible. Once the retrieval images are found for all the images in the query table, this information is displayed within the Generation.php file. When this file is executed, it will show the query and retrieval image side by side. Then I could quickly go through each pair and label the degree of quality. Once the quality was found, it would generate an excel file containing all the statistical information such as the query and retrieval image, Euclidean distance, Time for retrieval, and the degree of quality.

Chapter 7

Discussion and Summary

In this paper, we have presented a novel approach to represent novelty from video data effectively. This approach attempts to solve four main problems within this domain. The first problem has to do with summarizing meaningful information from video streams. Numerous work have been done to summarize an entire video, but this is meaningless since the events in a video may not be important. We have used a novelty detection system in video streams to detect only important events. We have shown in our results that by using our modified clustering algorithms, the summarizing do indeed provide meaningfull information.

Archiving is a major issue in any domain, especially with images and its annotations. Numerous systems exist today to automatically generate annotations based on the features in images. Also, images are annotated the old fashioned way by manually describing the images. Currently there are astronomical amounts of images and its annotations within the Internet today. The major problem is how this information is stored together. Annotations and images are usually stored separately with a link to connect between them. This causes numerous integrity problems when transfering to other locations with different schemes. This information is meaningless if the link between the two data sets are ever destroyed or lost. Our second contribution has shown that using JPEG 2000, we can solve these problems. This technology is utilized to effectively store both image and its information together for recovery in case of a failure.

Within a novel query engine, the retrieval is the most important part. Our third contribution is providing the most efficient methods to retrieve novel data using multiple indexing schemes for different types of data. We have surveyed numerous state-of-the-art indexing schemes, implemented each scheme, and compared and contrasted our findings for each scheme.

Our final contribution dealt with determining the quality of novelty data found from our novelty detection system to human perception. Our main problem was to determine how to measure human perception since we as humans have not yet been able to understand the human brain. We have devised a technique based on eye movements to generate similar

results from our novelty detection system and been able to give a comparative study between the two.

The following sections is an analysis of each contributions and its results. This is followed by opportunities for future work in all of these four areas.

7.1 Novelty Extraction

In our query engine system, we are using the VENUS system to detect novelty from video streams. This system attempts to mimic human perception by applying the technique called habituation. For example, if you walked into a busy terminal at an airport. If there was a group of kids playing and being loud, this would be the one of the first things you may notice. After awhile, you may get used to the sound and soon able to ignore it. Then suppose music started playing really loud from the intercom system. This would be very noticable at first, but will get accustom to it after some time. All these examples use the concept of habituation. New events such as the noise from the kids or music from the intercom may be interesting to you at first or novel, but you will eventually ignore it or have habituated to that event. This system may create numerous images of the same event as it habituates.

There may be numerous events within a single novel image. In our original design, we have extracted features from the full image (non-regions) to describe the event. But through numerous tests, we have determined that the features extracted do not explain a single event, rather multiple events occuring at the same time. Thus, we have extracted each novel region or connected component from the image, then extract features from the image. We have shown in our results that based on quality, regions provide better performance than non-regions.

We are currently extracting low level features which are Hue, Saturation, Value (HSV); Orientation, and Intensity. Hue is the amount of color that is perceived within the image. Saturation is based on the number of black and gray pixels found within the color aspects of the image. Value is the amount of brightness found in the image. Orientation is found by finding four 45 degree intervals to determine the number of edges in terms of pixels. Our last feature is intensity which is similar to brightness, but is based on a threshold to determine the pixel count. In our research, we have found the numerous video summarizing techniques that used Red, Green, and Blue features. However, RGB features are constricted since there can only be 256 values for each color. Where HSV has a larger range and also not soley based on whether one image has more red or not. We have tested both RGB and HSV and found the HSV provides better feature sets to work with.

Our clustering algorithm has been used to summarize these novel events. The clustering algorithm converts the image feature sets to a binary representation based on a threshold. The threshold is first determined by finding the average for each feature set. We have

discovered that some feature sets may cluster around the average. Based in our approach, we find the upper bound of this average to ensure this grouping is not split up. The binary values are clustered together and then the best representative images are found. Based on our results for both regions and non-regions, clustering has indeed performed better than a non-clustered data set.

7.2 Storage Using JPEG 2000

In theory, JPEG 2000 provides the utility to embed any type of information within the image. Also, a region can be extracted from the image easily. Before we could incorporate this technology in our query engine, we had to develop some tools for a proof of concept. Initially we modified a Kakadu [39] application to manually annotate an image. In this tool, we can open an image and select a region within the image itself. It will open a window where you can place your own annotations. This also improves annotation quality, since most annotations are based on the whole image. As in the same case in the previous section, the annotation describes the whole image rather than a single event. Thus, we are able to annotate both a specific region and the whole image separately. The information is embedded into the image in XML format. We decided to use this format for easy extractability purposes. By embedding information inside an image, we proved the first of three theories of this technology.

The second theory is to allow easy extractability of the information. We developed a tool located on a webserver where we can upload an annotated image where the server will automatically extract the information and place it within a database. This showed us that it is possible to extract any information from the image itself. Our final theory was to retrieve the region or region of interest (ROI) from the JPEG 2000 image. This applies to a major problem with traditional image formats. For satelite imagery, image sizes can be up to hundreds of gigabytes in size. It is impossible to send a single image through the internet. With our application we have show that it is possible to send just a region of the image back to the user. The user could then request more information and download the rest of the image or higher resolution without re-downloading the image again.

Our experiments on JPEG 2000 proved that this is a reliable technology to use for archiving purposes. We then applied our techniques in the applications to the novelty domain. In our query engine, we produce image information pertaining to region coordinates (bounding box), features, and cluster information. We then embed all of this using XML within a JPEG 2000 image. In the next section when we implement our indexing schemes, we can extract this information from the image to be indexed upon. Then we can retrieve the ROI for an image based on a query.

7.3 Indexing

In our system, we have proposed an approximate based indexing using clustered novel video data. We have shown through our results that in terms of speed and quality performs better than without using clustering. This framework was built on top of a R-tree implementation to handle its spatial capabilities. In order to determine if this indexing scheme was the best fit for our system, we surveyed 20 different state-of-the-art indexing schemes. After analysis of each indexing scheme, we narrowed down our survey to three indexing implementations that we felt was the most interesting and comparative to our study.

HDoV-tree

Our first indexing scheme we investigated and implemented is called the HDoV-tree designed by Shou et al [34]. This indexing scheme improved a user's point of view of 3D objects. Their indexing scheme is similar to a R-tree but uses two new concepts. The first concept is called the Level of Detail (LoD) which means the amount of detail a certain object should have based on a person's perspective. Closer objects will have a higher level of detail and farther objects otherwise. The second concept is called the Degree of Visibility (DoV) which means that objects behind other objects at a certain perspective in not important and should be ignored. This is a measurement that is calculated for each point of view that measures which objects should be visible at a certain perspective. These two concepts are very interesting and a different approach to view the data. Obviously our data set is not in 3D form, so we cannot use the exact methods they did. But we can use their ideas and develop creative approaches for our data set.

In our system, we have interpreted LoD to be an approximation of the images for a certain group. A R-tree is made up of subtrees where each subtree contains a recursive set of subtrees until it reaches the leaf nodes. For the entire R-tree, each parent will contain the best represetative image for images below it. This provides us with information about the underlying nodes of each sub-tree without traversing to the actual node.

Our main goal in this system is to retrieve similar images based on an image query. In order to incoporate this concept and the DoV, we also determine DoV for each query. DoV in our system is based on a reciprocal distance value between the query image and each image in the indexing scheme. The DoV is stored at each parent node which represents the aggregate of the reciprocal distance of all the underlying nodes. This value will give us a range between 0 and 1, which will then allow us to determine how deep we want to traverse the tree. If a certain node we see during a traversal is under a certain threshold, we can retrieve the LoD for this node and return it back to the system.

The main advantage to this indexing scheme is that the tree provides us with more information relating to the query engine system than a simple tool to retrieve information.

By having DoV and LoD, we can traverse the index with more efficiency. However, the theory behind the original HDoV theory has some issues. The overhead required to regenerated the DoV for every query is quite large and can increase retrieval time as seen in our comparative results. We have tried to handle this issue by caching the DoV values in case of repetitive queries. Another approach we have implemented that reduces the retrieval time is another interpretation of the DoV. This approach is based on the amount of coverage a single subtree has with respect to the entire data set. By knowing the visibility of data during the traversal, we can determine if we want to view a more general or exact similarities. This approach does produce better results than our previous results, but is further away from the original design since the DoV is performed once when the LoD is calculated. Even though the first approach did not perform as well, we are able to view the data at a whole new perspective.

P-tree

The second indexing scheme we reviewed is based on a Peer-2-Peer (P2P) network called P-trees designed by Crainiceanu et al [11]. Each peer is represented by a single value and its P-tree contains other peer references. The P-tree is based on a B-tree indexing scheme where partial P-trees sit at each peer. Its different from a B-tree in that each partial tree's root node is located at the leftmost node (For more of a detailed section of P-trees, refer to Chapter 1 on Indexing).

Initially this tree stucture was very difficult to understand since there were numerous new rules for this indexing scheme that was not clearly defined within the P-tree paper. Numerous assumptions were made in our design to mimic their system. These assumptions deal with the overlapping and repeating of peer nodes that occured in each P-tree.

Through our analysis of this algorithm, we did find some issues. For every new peer that joins the network, it must communicate to a certain node and then tell the rest of the nodes. Before anyone can insert the new peer, they must wait before the new peer has completed building its P-tree. This may cause deadlock between peer nodes which is a concern in this indexing scheme. Also, retrieval time is also based on network speed which is another overhead cost that is different from our original R-tree design. This network cost may result in poor retrieval performance time.

However, a great advantage to this indexing scheme is the entire data set does not have to exist in one location. The data can be stored at multiple locations and still be queried upon at these locations using this design. In our design, we felt that our clustered data set was a good fit for this indexing scheme. Each peer can contain a single cluster and be queried upon at any peer node. In order to do this, we had to modify the P-tree design. The original design contained a single value representing the peer. For each cluster, it contains the best representative image. This image values can be used to represent each peer within

the P-tree. In order to handle the cluster of images, we used our original R-tree to be stored at each peer. This indexing scheme introduces numerous opportunities and scalability for our system which improves the overall structure as a whole.

Singular Value Decomposition based indexing

Our final indexing scheme is based on Singular Value Decomposition (SVD) of multivariate data designed by Li et al [22]. The purpose of SVD is to decompose a large variable data set to a smaller data set with the same amount of variability. For a given matrix of data, SVD would decompose this matrix into three matrices called the left, singular, and right (4.1). In this scheme, the left matrix was used to define each attribute. Based on this left matrix, they created a suffix like tree where each attribute represented each level of the tree.

We felt that this scheme would be best suited for our data set since we have a multiple attribute feature set for a set of images. We also implemented SVD and organized the left matrix into a suffix like tree. Within the suffix like tree, each node is represented by a set of intervals. For example, if the values for the first attribute had the range of 0 to 1.5 for its left matrix values. In our implementation, we had three intervals where for this example the intervals would be 0 to 0.5, 0.5 to 1.0, and 1.0 to 1.5. The original implementation used four intervals. The number of intervals in each node has a significant impact of the quality of images returned. If the interval account was large, there would be quite a few "dangling pointers" to the image. This is because if you divide the root node m times for eight levels, each level would contain m^m nodes. Based on a certain path during the traversal, you may lead to a path which may not have a image at the end which is a problem. If you have too few intervals, then the pruning effect will not have a role. This is because all the images will be grouped together based on a certain interval. Based on our testings, we felt that three intervals was best for our data set. This interval can be changed for other types of data sets.

A major advantage using this indexing scheme is it does not need clustering to improve its results. It is an advantage to have more data in your data set than a smaller amount. This concept is a lot different from any other indexing scheme we studied. SVD works well when there is a large data set, since there are more images to compare each other too. Thus, clustering did not perform as well as other indexing schemes. Because the clustered data was a lower amount and did not have that many to compare it to, thus making the left matrix values very broad. Although, out of all the indexing schemes we studied, including the original R-tree, this indexing scheme performed the best when using a non-clustered data set.

7.4 Human Perception

Human perception was retrieved by using an eye tracker placed on each test subject that would watch a video on a 42 inch screen. An eye tracker is a device place on a person's head that will read the person's eye movements. For each frame within the video, fixations depicted as dots of where the person was looking at. By taking a slice from the video stream and place this set of frames together, we can cluster these fixations together and create a mask similar to the one applied to VENUS. Based on the cluster of fixations, we find the centroid of fixation and use that video frame to represent this cluster. The mask is placed on top of the centroid video frame which creates similar novel images like VENUS.

The number of frames in a video taken for a slice is based on a pre-determined threshold. If this threshold was large, we would get a lot of fixations which may produce a large novel region for a single frame. This may cause problems since some the novel region may contain multiple events. Other problems may have to do with the extreme fixations from the beginning and ending video frames of a slice may be completely different. If the threshold was set really small, then there would not be as many fixations. This would cause small novel regions which may not depict anything at all. We were able to obtain a good threshold value by training the application on multiple values and measuring the quality of the novel images.

In our comparative study between human perception and machine novelty, we used our clustering techniques described in this paper. Representative images were found for both data sets and compared. In our initial design, we developed two different metrics based on region and location. Region is based on the feature set of two images and location is based on the similarity of the coordinates between two events. We used both metrics together to compare our results. However, since our novel query engine is using Euclidean distance to measure between the query and retrieval image. We decided to base the comparison on this type of distance also. Also, since we are trying to determine how far or close algorithmic novelty detection is from human perception, we using machine based comparisons. This means that for each representative image found by the machine novelty detection system (VENUS), we found the closest representative image from the human perception data set. Based on our results, we found that even though VENUS is mimicking a natural trait in humans, the degree of quality was poor. It is interesting to note that in machine novelty detection, it is able to detect multiple events at the same time. Where in humans, they can only detect a single event at the same time. At this aspect, machine novelty detection performed far better than human perception.

We have also extracted regions from the human novelty data. Regions can occur when the threshold for slicing is really large and multiple events occur during a single slice. These regions were extracted for each video and also showed the same results as previous tests

that regions perform better than non-regions.

7.5 Future Work

The contributions made within this paper provide numerous possibilities of advancements in all areas of meaningful summarization, storage, retrieval, and human perception that are out of scope for this project. Our system utilized VENUS to provide novelty data from video streams. Further study on different novelty detection systems would be very useful in this system. Other systems such as developed by Diehl and Hampshire [14] that clasify objects based on previously labeled objects. By having a comparative study on different novelty detection systems could then plug into this system which may produce better results than it does today. In our framework, we using low level features, other opportunites is the ability to extract more complex objects such as scenes or using surrounding audio in a video.

The tools we developed for JPEG 2000 is simply a stepping stone for an infinite number of possibilities. Such as embeding other types of multimedia data like other images, audio, video, etc. Based on JPEG 2000 theory, it is possible to stream the image to the client from a web server. This could improve performance time when extracting regions from the image. An investigation of the JPEG 2000 compression algorithm can be done to investigate the amount of improvement between traditional image formats.

Currently there is no standard for clustering or indexing in any domain, thus there are numerous applications and different algorithms to study. The P-tree that we developed has numerous improvements that can be done. In the P-tree original design by Crainiceanu et al [11] does not simulate real P2P networks. In this design, each peer is represented by a certain value. In a P2P system, there can be multiple values within each peer. A possible opportunity is to redesign the P-tree algorithm and incorporate ranges for each node. Also, other types of indexing schemes can be explored such as the KDB-tree which handles multivalue data sets.

Our human perception novelty detection system is based on an eye tracker to detect fixations. Numerous advancements have been made on this type of technology to obtain more efficient results. Similar to the study of different machine novelty detection systems, but for human perception. The most efficient system could be used and the compared to the machine novelty detection system using our framework.

Bibliography

[1] Support vector machines. http://www.statsoftinc.com/textbook/stsvm.html.

[2] Lee A. Virtualdub. http://www.virtualdub.org.

[3] Tentler A., Vaingankar V., Gaborski R., and Teredesai A. Event detection in video sequences of natural scenes. In *In IEEE Western New York Image Processing Workshop*, Rochester, New York, 2003.

[4] James Allan, Courtney Wade, and Alvaro Bolivar. Retrieval and novelty detection at the sentence level. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 314–321, New York, NY, USA, 2003. ACM Press.

[5] Noga Alon, Raphy Yuster, and Uri Zwick. Color-coding: a new method for finding simple paths, cycles and other small subgraphs within large graphs. *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, ACM Press(New York, NY, USA), 1994. 326–335.

[6] Michael A. Arbib. *The Handbook of Brain Theory and Neural Networks*. MIT Press, Cambridge, MA, USA, 1995.

[7] Lars Arge, Mark de Berg, Herman J. Haverkort, and Ke Yi. The priority r-tree: a practically efficient and worst-case optimal r-tree. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 347–358, New York, NY, USA, 2004. ACM Press.

[8] D. A. Beckley, M. W. Evens, and V. K. Raman. Multikey retrieval from k-d trees and quad-trees. In *SIGMOD '85: Proceedings of the 1985 ACM SIGMOD international conference on Management of data*, pages 291–301, New York, NY, USA, 1985. ACM Press.

[9] Michael C. Burl. Mining patterns of activity from video data. In Michael W. Berry, Umeshwar Dayal, Chandrika Kamath, and David B. Skillicorn, editors, *SDM*. SIAM, 2004.

[10] Matthew Cooper, Jonathan Foote, Andreas Girgensohn, and Lynn Wilcox. Temporal event clustering for digital photo collections. In *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*, pages 364–373, New York, NY, USA, 2003. ACM Press.

[11] Adina Crainiceanu, Prakash Linga, Johannes Gehrke, and Jayavel Shanmugasundaram. P-tree: a p2p index for resource discovery applications. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 390–391, New York, NY, USA, 2004. ACM Press.

[12] Paul A. Crook, Stephen Marsland, Gillian Hayes, and Ulrich Nehmzow. A tale of two filters - on-line novelty detection. In *ICRA*, pages 3894–3899. IEEE, 2002.

[13] Dasgupta, Dipanker, and Stephanie Forrest. Novelty detection in time series data using ideas from immunology. In *Proceedings of the 5th International Conference on Intelligent Systems*, Reno, Nevada, USA, 1996.

[14] Christopher P. Diehl and John B. Hampshire. Real-time object classification and novelty detection for collaborative video surveillance. In *Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on*, volume 3, pages 2620 – 2625, Honolulu, HI, USA, May 2002.

[15] David Eppstein. Subgraph isomorphism in planar graphs and related problems. In *SODA '95: Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 632–640, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

[16] Roger S. Gaborski, Vishal S. Vaingankar, Vineet Chaoji, and Ankur Teredesai. Venus: A system for novelty detection in video streams with learning. In Valerie Barr and Zdravko Markov, editors, *FLAIRS Conference*. AAAI Press, 2004.

[17] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. pages 599–609, 1988.

[18] Eric Haseltine. User-centered design for kdd. In *KDD '04: Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1–1, New York, NY, USA, 2004. ACM Press.

[19] Eamonn Keogh, Stefano Lonardi, and Bill 'Yuan chi' Chiu. Finding surprising patterns in a time series database in linear time and space, 2002.

[20] David G. Kirkpatrick and Pavol Hell. On the completeness of a generalized matching problem. In *STOC '78: Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 240–245, New York, NY, USA, 1978. ACM Press.

[21] T. Kohonen. *Self-organization and associative memory: 3rd edition*. Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[22] Chuanjun Li, Gaurav Pradhan, S. Q. Zheng, and B. Prabhakaran. Indexing of variable length multi-attribute motion data. In *MMDB '04: Proceedings of the 2nd ACM international workshop on Multimedia databases*, pages 75–84, New York, NY, USA, 2004. ACM Press.

[23] Jessica Lin, Eamonn Keogh, and Wagner Truppel. Clustering of streaming time series is meaningless. In *DMKD '03: Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 56–65, New York, NY, USA, 2003. ACM Press.

[24] Corbetta M, Akbudak E, Conturo TE, Snyder AZ, Ollinger JM, Drury HA, Linenweber MR, Petersen SE, Raichle ME, Van Essen DC, and Shulman GL. A common network of functional areas for attention and eye movements. In *Neuron*, volume 21, pages 761–773, 1998.

[25] Junshui Ma and Simon Perkins. Online novelty detection on temporal sequences. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 613–618, New York, NY, USA, 2003. ACM Press.

[26] Markos Markou and Sameer Singh. Novelty detection: a reviewpart 1: statistical approaches. *Signal Process.*, 83(12):2481–2497, 2003.

[27] Sougata Mukherjea, Kyoji Hirata, and Yoshinori Hara. Using clustering and visualization for refining the results of a www image search engine. In *NPIV '98: Proceedings of the 1998 workshop on New paradigms in information visualization and manipulation*, pages 29–35, New York, NY, USA, 1998. ACM Press.

[28] Alexandre Nairac, Timothy A. Corbett-Clark, Ruth Ripley, Neil W. Townsend, and Lionel Tarassenko. Choosing an appropriate model for novelty detection. In *Artificial Neural Networks, Fifth International Conference on (Conf. Publ. No. 440)*, pages 117–112, Cambridge, July 1997.

[29] Jung-Hwan Oh, JeongKyu Lee, and Sanjaykumar Kote. Real time video data mining for surveillance video streams. In Kyu-Young Whang, Jongwoo Jeon, Kyuseok Shim, and Jaideep Srivastava, editors, *PAKDD*, volume 2637 of *Lecture Notes in Computer Science*, pages 222–233. Springer, 2003.

[30] Jung-Hwan Oh, JeongKyu Lee, Sanjaykumar Kote, and Babitha Bandi. Multimedia data mining framework for raw video sequences. In Osmar R. Zaïane,

Simeon J. Simoff, and Chabane Djeraba, editors, *Revised Papers from MDM/KDD and PAKDD/KDMCD*, volume 2797 of *Lecture Notes in Computer Science*, pages 18–35. Springer, 2002.

[31] Miles Ohlrich, Carl Ebeling, Eka Ginting, and Lisa Sather. Subgemini: identifying subcircuits using a fast subgraph isomorphism algorithm. In *DAC '93: Proceedings of the 30th international conference on Design automation*, pages 31–37, New York, NY, USA, 1993. ACM Press.

[32] Jia-Yu Pan and Christos Faloutsos. Videocube: A novel tool for video mining and classification. In *ICADL '02: Proceedings of the 5th International Conference on Asian Digital Libraries*, pages 194–205, London, UK, 2002. Springer-Verlag.

[33] U. Nehmzow S. Marsland and J. Shapiro. Detecting novel features of an environment using habituation. In *Proc. Simulation of Adaptive Behavior*, pages 189–198. MIT Press, January 2000. ISBN 0262632004.

[34] Lidan Shou, Zhiyong Huang, and Kian-Lee Tan. Hdov-tree: The structure, the storage, the speed. In Umeshwar Dayal, Krithi Ramamritham, and T. M. Vijayaraman, editors, *ICDE*, pages 557–568. IEEE Computer Society, 2003.

[35] D. A. T.; Siddle, M. Kuiack, and S. B. Kroese. The orienting reflex. *In Physiological Correlates of Human Behaviour*, pages 149–170, 1983.

[36] I. Skiljan. Irfanview. Vienna University of Technology, 2004.

[37] Chris Stauffer and W. Eric L. Grimson. Learning patterns of activity using real-time tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):747–757, 2000.

[38] L. Tarassenko. Novelty detection for the identification of masses in mammograms. In *Proceedings of the 4th IEE International Conference on Artificial Neural Networks*, volume 4, pages 442–447, Cambridge, UK, 1995.

[39] D. Taubman. Kakadu: A foundation for jpeg 2000 applications, 2001. The University of New South Wales, Sydney.

[40] C. Tong and V. Svetnik. Novelty detection in mass spectral data using a support vector machine method, 2002. Computing Science and Statistics.

[41] Shingo Uchihashi, Jonathan Foote, Andreas Girgensohn, and John Boreczky. Video manga: generating semantically meaningful video summaries. In *MULTIMEDIA '99: Proceedings of the seventh ACM international conference on Multimedia (Part 1)*, pages 383–392, New York, NY, USA, 1999. ACM Press.

[42] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.

[43] R.R. Wang and T.S. Huang. A framework of human motion tracking and event detection for video indexing and mining. In *Proc. DIMACS Workshop Video Mining*, 2002.

[44] M. Windhouwer, R. Zwol, H. Blok, W. Jonker, M. Kersten, and P. Apers. Content-based video indexing for the support of digital library search. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, pages 494–495, Washington, DC, USA, 2002. IEEE Computer Society.

[45] Lexing Xie, Shih-Fu Chang, Ajay Divakaran, and Huifang Sun. Unsupervised mining of statistical temporal structures in video. In A. Rosenfeld, D. Doremann, and D. Dementhon, editors, *Video Mining*, chapter 10. Kluwer Academic Publishers, 2003.

[46] Kenji Yamanishi and Jun ichi Takeuchi. A unifying framework for detecting outliers and change points from non-stationary time series data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 676–681, New York, NY, USA, 2002. ACM Press.

[47] Y. Yang, J. Zhang, J. Carbonell, and C. Jin. Topic-conditioned novelty detection, 2002.

[48] Dave Yanofsky. Quick facts, 2003. http://www.justthink.org/resources/facts.html.

[49] HongJiang Zhang, Atreyi Kankanhalli, and Stephen W. Smoliar. Automatic partitioning of full-motion video. *Multimedia Syst.*, 1(1):10–28, 1993.

[50] Xingquan Zhu, Jianping Fan, Ahmed K. Elmagarmid, and Xindong Wu. Hierarchical video content description and summarization using unified semantic and visual similarity. *Multimedia Syst.*, 9(1):31–53, 2003.

[51] Xingquan Zhu, Jianping Fan, Mohand-Said Hacid, and Ahmed K. Elmagarmid. Classminer: mining medical video for scalable skimming and summarization. In *MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia*, pages 79–80, New York, NY, USA, 2002. ACM Press.