An Application Specific Low Bit-Rate Video Compression System Geared Towards Vehicle Tracking

Ryan David Spicer

Submitted in fulfilment of the academic requirements for the degree of Master of Science in Engineering in the School of Electrical and Electronic Engineering, University of Natal, Durban.

PREFACE

The author carried out the work described in this thesis during the period February 2000 to September 2002 under the supervision of Professor R. Peplow in the School of Electrical and Electronic Engineering, University of Natal, Durban.

The author certifies that this thesis represents his own original work, except where specifically indicated to the contrary in the text, and that it has not been submitted to any other university for degree purposes.

As the candidate's supervisor I have/have not approved this thesis/dissertation for submission.

Signed : Name : Date :

ACKNOWLEDGEMENTS

Thanks go to Professor Peplow for his guidance over the last two years and also for his motivational discussions when they were needed most. I am also very grateful for his endless patience whilst editing this thesis. Thanks are also owed to Mr. McDonald for his understanding and his organisation during the course of this work.

A big thank-you to ARMSCOR for their sponsorship of this work. Without their support, this work would not have been possible.

My fellow postgraduate colleagues have made the past two years extremely enjoyable. For their presence, I thank them.

I wish to thank my family for their love, support and understanding over the last two years. Finally, I would like to extend a very special note of thanks to my Fiancée and soul mate Phillipa Martin for her love and encouragement, especially over the last trying year. Her ability to keep me focussed on the goal has culminated in the presentation of this work. For this, I am greatly indebted.

ABSTRACT

The ability to communicate over a low bit-rate transmission channel has become the order of the day. In the past, transmitted data over a low bit-rate transmission channel, such as a wireless link, has typically been reserved for speech and data. However, there is currently a great deal of interest being shown in the ability to transmit streaming video over such a link. These transmission channels are generally bandwidth limited hence bit-rates need to be low. Video on the other hand requires large amounts of bandwidth for real-time streaming applications. Existing Video Compression standards such as MPEG-1/2 have succeeded in reducing the bandwidth required for transmission by exploiting redundant video information in both the spatial and temporal domains. However such compression systems are geared towards general applications hence they tend not to be suitable for low bit-rate applications.

The objective of this work is to implement such a system. Following an investigation in the field of video compression, existing techniques have been adapted and integrated into an application specific low bit-rate video compression system. The implemented system is application specific as it has been designed to track vehicles of reasonable size within an otherwise static scene. Low bit-rate video is achieved by separating a video scene into two areas of interest, namely the background scene and objects that move with reference to this background. Once the background has been compressed and transmitted to the decoder, the only data that is subsequently transmitted is that that has resulted from the segmentation and tracking of vehicles within the scene. This data is normally small in comparison with that of the background scene and therefore by only updating the background periodically, the resulting average output bit-rate is low.

The implemented system is divided into two parts, namely a still image encoder and decoder based on a Variable Block-Size Discrete Cosine Transform, and a context-specific encoder and decoder that tracks vehicles in motion within a video scene. The encoder system has been implemented on the Philips TriMedia TM-1300 digital signal processor (DSP). The encoder is able to capture streaming video, compress individual video frames as well as track objects in motion within a video scene. The decoder on the other hand has been implemented on the host PC in which the TriMedia DSP is plugged. A graphic user interface allows a system operator to control the compression system by configuring various compression variables. For demonstration purposes, the host PC displays the decoded video stream as well as calculated rate metrics such as peak signal to noise ratio and resultant bit-rate.

The implementation of the compression system is described whilst incorporating application examples and results. Conclusions are drawn and suggestions for further improvement are offered.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
ADC	Analogue-to-Digital Converter
B- Frame	Bi-Directional Frame
Bpp	Bits per Pixel
CIF	Common Intermediate Format
CR	Compression ratio
DCT	Discrete Cosine Transform
DSP	Digital Signal Processor
GUI	Graphic User Interface
HDTV	High Definition Television
HVS	Human Visual System
IDCT	Inverse Discrete Cosine Transform
I-Frame	Intra Frame
ISO	International Organisation for Standardisation
JPEG	Joint Photographic Experts Group
KLT	Karhunen-Loeve Transform
MAD	Mean Absolute Difference
MJPEG	Motion JPEG
MPEG	Moving Pictures Expert Group
MSE	Mean Squared Error
MSVC++	Microsoft Visual C++
NTSC	National Television System Committee
PAL	Phase Alternation by Line
P-Frame	Predicted Frame
PSNR	Peak-Signal-to-Noise Ratio
Qfactor	Quality Factor
RGB	Red, Green and Blue
RLD	Run-Length Decode
RLE	Run-Length Encode
SAD	Sum of the Absolute Differences
SDE	Software Development Environment
SECAM	Sequential Colour with Memory
U	Chrominance Colour Component
V	Chrominance Colour Component
VF	Video Frame
VQ	Vector quantisation
VS	Video Sequence
Y	Luminance Colour Component
YUV	Additive colour system, Y = luminance, U and V = chrominance

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Over the last two decades, there has been an exponential growth in the processing power of available microprocessors that has subsequently resulted in applications involving the transmission and storage of data, more specifically digital video, growing at a rapid rate. Typical video processing applications that are continually gaining in popularity include high definition television (HDTV), video conferencing, digital versatile disks (DVD), video over the LAN as well as over wireless links. However in its raw format, digital video generates huge volumes of data. For example, raw PAL video requires approximately 110 giga-bytes of storage space per hour and would thus require extremely high transmission bandwidths. In order to reduce the size of this data so that it is more manageable, different forms of compression can be applied whereby the data stream is encoded before initial storage or transmission. With this in mind, the goal of any video compression algorithm is to maximise the compression ratio[1] of a video stream thereby minimising the required transmission bandwidth, while at the same time minimising the decrease in fidelity of the decoded video stream that results as a consequence of compression.

Considering a video stream comprises a sequence of still images, traditional video compression systems take advantage of the redundancies that are exhibited in video in both the spatial[2] and temporal[3] domains. The general concept behind a spatial domain compression algorithm is to create a statistical model of an image from application of specific equations that have been derived from information theory. The statistical characteristics themselves are then manipulated ultimately resulting in less space being required to store the image compared to that that would have been required had the image been stored in its original format. The majority of existing spatial domain compression systems are based on waveform encoding techniques. Some of the more commonly used waveform based encoding techniques include the popular discrete cosine transform (DCT), wavelets and fractals.

The compression of video on the other hand is achieved from a combination of spatial domain coding as well as manipulating the similarities that exist between consecutive video frames. One of the more common techniques used to reduce temporal redundancies is that of motion compensation, typically by means of block matching [Jain81]. This process involves sub-dividing video frames into small blocks and attempting to match corresponding similar blocks between consecutive frames. Block matching effectively results in reduced output bit-rates[4]. Several international video compression standards are based on block matching techniques. These include the Moving Pictures Expert Group (MPEG) 1, 2 [MPEG-01] and 4 [MPEG-4-01], the H.263 video conferencing standard [H.263] as well as its predecessor H.261. The general operation of these encoding techniques is discussed in greater detail in the following chapter.

[1] Compression ratios indicate the ratio between the original input data and the compressed output data sizes. A compression ratio of 2:1 indicates that the output data is half the size of the original input data.

[2] The term spatial refers to data belonging to a single image.

[3] Temporal refers to a sequence of still images constituting a video stream.

[4] Measurement index that specifies the number of bits a video coding system can output per second.

Although these standards operate well and have established track records, they prove inefficient when applied to applications that require very low output bit-rates. Research over the last few years has indicated that instead of coding a video scene as a singular unit, the separation of the content of a video scene into more meaningful entities of varying importance and the subsequent coding thereof, assists in achieving very low output bit-rates [Ebrah00], [Han98], [MPEG-4-01] and [Symes98]. It is this concept of separating a video scene into different entities that a significant portion of the work undertaken in this dissertation concentrates on. However before delving into low-level detail, some fundamental concepts require introduction. The following sections describe what is meant by the term application specific, what objects are, what video is and how it can be compressed with regard to the human visual system (HVS).

1.1 What is Application Specific?

An application specific system, as its name suggests, can be defined as a system that is designed for a specific task. In other words, within its intended application, the performance and operation of the system would be predictable within certain limits. The purpose of an application specific system is to achieve extraordinary results that a general purpose system would not be able to achieve. Such results could include output bit-rate, compression, decoded clarity or even algorithmic speed.

Examining existing systems, MPEG-1 is one of the most common video compression systems that can be applied to wide variety video streams and can therefore be classified as a general purpose compression system. The closest MPEG-1 comes to being application specific is that it requires input video to be a certain resolution and the standard specifies that the output bit-rate should be approximately 1.5 Mbits/second. H.263 is an example of a video standard that is more application specific than MPEG-1. The intended use of H.263 is for video conferencing applications, although it is not strictly the case now days. Video conferencing generally translates to a head and shoulders environment and thus constitutes low activity. The result is that within this situation, H.263 can guarantee low output bit-rates and is therefore suitable for transmission via telephone networks.

An application specific video compression system is proposed and implemented within this dissertation. The system can be categorised as application specific for two main reasons. Firstly, the intended use of the system is to track vehicles within a generally low activity scene and as a consequence, extremely low output bit-rates are achieved. Secondly, the implemented algorithms are optimised towards the encoding and decoding of rectangular shaped objects i.e. vehicles. Within the restrictions of its intended purpose, the goals of the system were achieved, however the same cannot be said when applied to situations that do not adhere to the specific requirements of the system. The performance of the implemented application specific video compression system is analysed in conjunction with obtained results in Chapter 5.

1.2 What are Objects?

Objects are identified 'things' within a video scene. The concept of identifying objects within a video scene and manipulating them individually is the subject of much research at present due to the distinct advantages that it offers. One such advantage is that the coding of objects rather than a complete video scene itself is more likely to result in lower output bit-rates. The reason for this is that objects are generally not as large as the complete video scene and would therefore require fewer bits for coding.

Secondly, the coding of objects as separate entities effectively preserves their fidelity when decoded as their respective coding models are dynamically adapted as the objects change in time.

The most recent video compression standard is that of MPEG-4. Having been officially standardised in 1999, the objective of the MPEG-4 [MPEG-4-01] standard is to provide reasonable quality video at very low bit-rates and very high quality at bit-rates comparable to that of its MPEG predecessors. MPEG-4 analyses a video scene and identifies areas of both high and low motion activity. The identified areas are then classified as either objects in motion or stationary objects, with each object being represented by its shape, motion and texture (colour). The three parameters of each object are then encoded and transmitted to the respective decoder. The 'tracking' of the objects in motion with respect to stationary objects, such as background scenes, allows for very low output bit-rates as stationary objects do not change in shape, motion and texture as frequently as objects in motion. Thus stationary objects only require updating in terms of re-encoding when a change is detected in at least one of its three defining parameters. Consequently, the efficiency and accuracy of the implemented extraction technique within a content-based coding system combined with its precision in terms of estimating the motion of moving objects within a video scene plays a crucial role in the characterisation of both the system's transmission rate and fidelity of the decoded video stream.

General motion estimation techniques as well as commonly utilised extraction (segmentation) algorithms are described in greater detail in Chapter 2.

1.3 What is Video?

Before describing various video compression techniques, an understanding of what video is and how it is obtained is presented. These described fundamentals are then elaborated upon in the following chapter as it provides an overview of the state of the art of image and video technology.

Video can be thought of as a series of still image samples (snapshots) taken over a period of time that are presented as a single continuous flow. By displaying these singular images at a continuous, rapid rate, the human viewer perceives the images in context with past and future images; hence the scene appears continuous in time. The following sections describe the different formats and types of video that are in use today.

1.3.1 Analogue Video

Unlike a Cinefilm camera that samples images in time by means of light sensitive film, video cameras use an electronic sensor to detect light intensities within a target scene [Sound01]. An electronic sensor is a two dimensional (2D) grid of photo-diodes. Each photo-diode within the sensor grid converts the number of photons striking it into a proportional voltage potential. The charge-coupled device (CCD) is a commonly used electronic sensor in video cameras available today. Traditional video cameras are analogue based systems in that the voltage potential of each photo-diode along a horizontal line is sampled and converted to an analogue waveform. The manner in which the potential of each photo-diode is sampled is termed *raster-scanning* [Lawyer97]. Raster scanning can be broken into two further categories: progressive and interlaced [Thomp98].

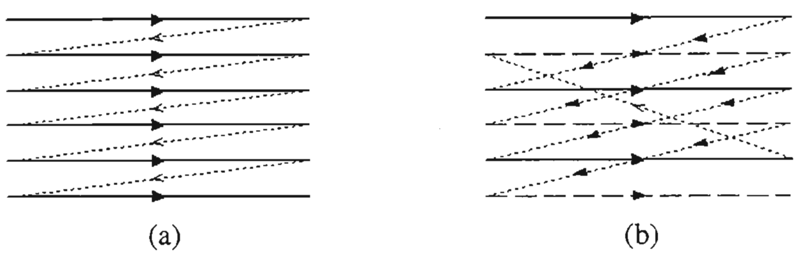


Figure 1-1: Two categories of raster scanning, (a) progressive scanning and (b) interlaced scanning.

Progressive scanning is the process where each row within the sensor grid is sampled horizontally from a left to right direction, from the top of the grid to the bottom as illustrated in Figure 1-1(a). Interlaced scanning divides the sensor lines into odd and even scan lines termed fields. Scanning commences along the odd field where all the odd lines are sampled from left to right along each row, from the top of the grid towards the bottom. Once the odd field has been scanned, the even scan lines are then sampled in a similar fashion. The process of interlaced scanning is illustrated in Figure 1-1(b).

There are three general formats that are implemented in television systems throughout the world today [Li00]. These include the National Television System Committee (NTSC) video format that specifies a total of 525 scan lines per video frame at a frame rate of 30 frames/second. NTSC employs interlaced scanning; hence each video frame is divided into two fields comprising 262.5 lines/field. NTSC is utilised in the U.S.A, Japan and other countries. The second commonly found analogue video format is that of the Phase Alternation by Line (PAL) video format. PAL specifies a total of 625 scan lines per frame at a frame rate of 25 frames/second. PAL also employs interlaced scanning hence each frame is constructed from 2 fields comprising 312.5 lines/field. PAL is used in Europe as well as some African countries including South Africa. The third video format is the Sequential Couleur Avec Memoire or Sequential Colour with Memory (SECAM). SECAM specifies a total of 625 scan lines per second at a frame rate of 25 frames/second. Although SECAM uses the same transmission bandwidth as PAL, it transmits the colour information sequentially unlike PAL. SECAM has its origins in France.

1.3.2 Digital Video

In order to process video on a computer for example, the output video signal from an analogue video camera has to be converted to digital format. There are two possible means of digitising video. In the case of a computer, a video capture card can be used to receive an analogue video signal and then convert this signal to digital format by means of an analogue-to-digital converter (ADC). This digitised information is then ready for processing. The other alternative of digitising video is at the video camera itself. Instead of sampling each line of photo-diodes within the 2D sensor grid and representing the output potential by an analogue waveform, an onboard ADC converts each photo-diode's potential to a physical number. As in the case of analogue video, digital video is scanned in either progressive or interlaced format. An onboard computer then processes the 2D array of converted values further and the resultant data can either be stored on digital videotape or viewed on a connected digital television monitor.

A single digital video frame can be thought of as a 2D array of digitised elements called pixels. The pixels are arranged within horizontal rows, with one row beneath the other. The mentioned analogue video standards also define their digital counterparts. According to the format standards for digital television specified by the Consultative Committee for International Radio CCIR-601 [Li00], each digital NTSC video frame comprises 704x480 (columns x rows) pixels with the video sampled at a frame rate of 30 frames/second. With regard to PAL as well as SECAM, the CCIR-601 specifies that each digital video frame comprises 704x576 pixels with the video sampled at a frame rate of 25 frames/second. In all three cases, the sampled digital video is in interlaced format. Table 1-1 lists the more common digital resolutions that are used in both broadcast and videoconferencing video systems [Digital01].

Table 1-1: Common digital video resolutions.

Name	Acronym	Horizontal	Vertical
	For Video Broadcasting		
National Television System Committee	**full – NTSC**	704	480
	half – NTSC	352	480
Phase Alternation by Line	**full – PAL**	704	576
	half – PAL	352	576
Sequential Colour with Memory	**full – SECAM**	704	576
	half – SECAM	352	576
Standard Interchange Format	**NTSC – SIF**	352	240
	PAL – SIF	352	288
	SECAM - SIF	352	288
	For Videoconferencing		
Common Intermediate Format (defined for the H.261 and H.263 standards)	**QCIF**	176	144
	CIF	352	288
	4CIF	704	576
	16CIF	1408	1152

1.3.3 Colour Spaces

With regard to the HVS, research indicates that there are two basic methods of producing colours that are recognisable by human beings with normal eyesight [Penne93]. These methods include additive and subtractive colour. Additive colour is created by adding together the different colours that are emitted by active light sources, resulting in a new perceived colour. Subtractive colour on the other hand is created by selectively absorbing the wavelengths of colours emitted by light sources resulting in the wavelength of the new perceived colour remaining. The most common additive colour system uses the three primary colours red, green and blue (RGB) while a common subtractive colour system uses the colours cyan, magenta and yellow (CMY).

Research in the field of colour reproduction and perception with regard to digital imaging and video has resulted in the creation of several different colour representations other than the RGB and CMY colour spaces [Bourke94]. Experimentation has revealed that a more accurate model of the HVS with regard to the perception of colour is one that is based on brightness, hue and saturation. Brightness describes the intensity or luminance of the light source, i.e. whether it is light, dark or a shade in between. Hue describes the actual colour of the light source, for example red, green, purple etc, while saturation describes the purity of the colour, i.e. if it is a strong colour, pastel or nearly white in perception. The hue saturation intensity (HSI) model is one such typical model. Another model that is closely related to the HSI colour space is the YUV colour space. The Y component of the YUV colour space models the luminance of an image, in other words it provides a greyscale (monochrome) representation of a colour image. The U and V components model two chrominance (colour) components of an image. Considering the two chrominance components are real values that have a range of [0, 1], the missing third chrominance component can be evaluated according to:

$$X = 1 - U - V \tag{1-1}$$

Depending on the nature of a particular application, it is possible that an image presented in the RGB colour space may require processing in the YUV colour space or vice versa. In this case a set of linear equations have been derived in order to perform such a transformation. Equations (1-2) and (1-3) provide the transformation relationship between the two colour spaces:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.148 & -0.289 & 0.437 \\ 0.615 & -0.515 & -0.10 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{1-2}$$

where R, G, B are real values in the range (ε) [0, 1], and

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.000 & 1.140 \\ 1 & -0.394 & -0.581 \\ 1 & -2.028 & 0.000 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix} \tag{1-3}$$

where Y ε [0, 1] and U, V ε [-0.5, 0.5]

Another commonly used colour space is YC_bC_r. This space was used extensively during the development of the Joint Photographic Experts Group (JPEG) [JPEG01] still image compression standard. Appendix A provides the transformation relationships of the above mentioned colour spaces as well as the CIELAB colour space.

1.3.4 Transmission Formats

There are three general formats in which video is transmitted [Li00]. These include:

- **Component video:** each primary colour component is transmitted from the video source as a separate video signal. For example, each of the three RGB or YUV colour components is transmitted individually. Although this type of video results in very high quality colour reproduction, it requires large amounts of bandwidth and good synchronisation between the three components.
- **Composite video:** (also known as CVBS video) all three-colour components are mixed together into a single carrier wave. The reduction in the required bandwidth is achieved at the expense of interference between the mixed signals resulting in a degradation of video fidelity. This is the most common form of commercially available video and is typically transferred between a video source and receiver by means of a common RCA cable.
- **S –Video:** (also known as Super Video) uses two lines for the transmission of video, one for the luminance component and the second for the composite chrominance components. S-Video is generally used in high quality video apparatus.

1.4 The Human Visual System

Considering that humans are the ultimate viewers and critics of a decompressed image or video stream, an understanding of the characteristics of the HVS is required. Referring to Figure 1-2, [Penne93] provides a plot of the response of the human eye to threshold intensity changes at different spatial frequencies. Spatial frequency refers to the measurement of the human eye's perception of a change in colour over varying distances. Spatial frequency in cycles/degree is calculated, according to [Gerf96]:

$$\omega = \frac{\pi}{180}.d.v \qquad (1\text{-}4)$$

where d is the viewing distance and v is the changing frequency of the viewed colour components.

The graph illustrated in Figure 1-2 represents the condition that a human eye can detect changes in colour as a function of spatial frequency.

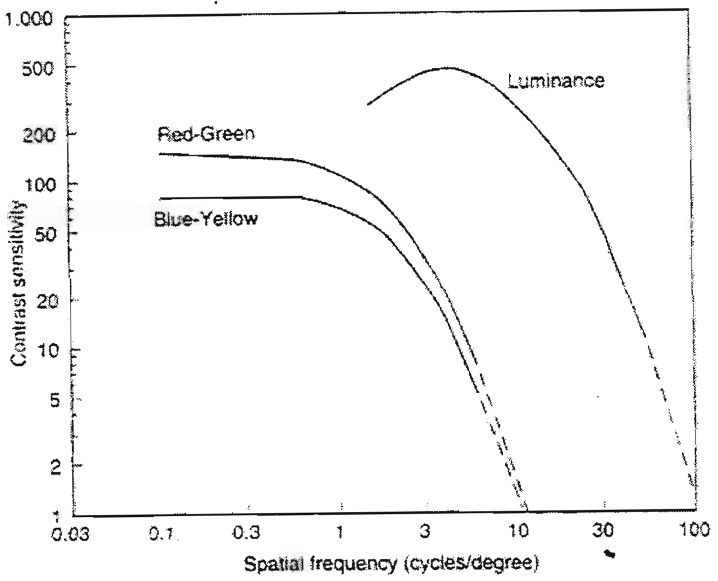


Figure 1-2: Sensitivity of the eye to luminance and chrominance spatial frequency changes [Penne93].

From the graph it is noted that chrominance curves decrease significantly before the luminance curve starts to decrease. What this means is that the human eye is far more sensitive to changes in luminance than it is to changes in chrominance. This property of the HVS provides a very important basis in terms of digital colour imaging and compression. From this property, efficient colour compression systems take advantage of the HVS by coding colour information into a smaller number of bits compared to those needed to represent luminance. Thus it is commonly found that compression systems operate in the luminance-chrominance colour space rather than the RGB or CMY colour spaces.

1.5 Colour Sub-Sampling

Based on the HVS' insensitivity to high frequency changes in chrominance compared to luminance, one of the most common methods of reducing the amount of chrominance data to be coded is that of sub-sampling. Sub-sampling is often, but not always, employed during the process of converting analogue video to digital video. With regard to a colour image, sub-sampling is the process of sampling the chrominance components at a lower rate than that of the luminance components. By doing so, fewer chrominance pixels result compared to the number of luminance pixels. The omission of this '*redundant*' chrominance data within the sampled image assists coding efficiency, as there are fewer pixels in total to encode. There are four generally accepted colour sub-sampling formats [Li00]. Figure 1-3 illustrates these formats with respect to the YUV colour space.

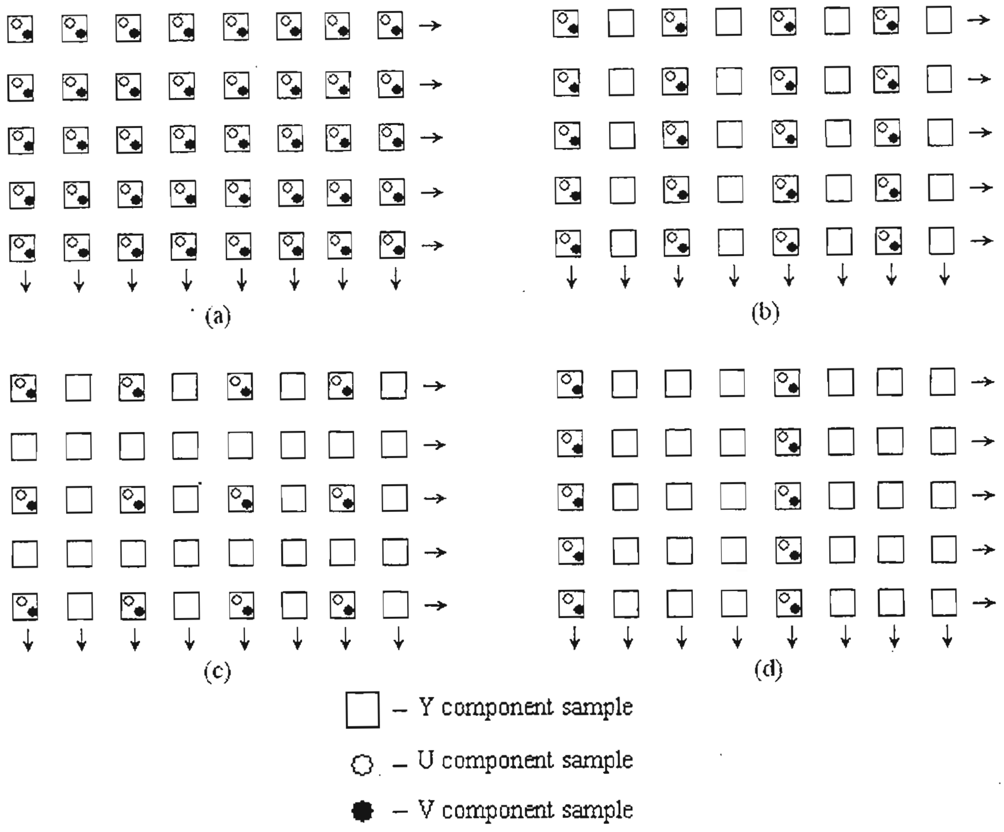


Figure 1-3: Colour sub-sampling picture formats with the appropriate legend indicated. (a) is 4:4:4 sub-sampling, (b) is 4:2:2 sub-sampling, (c) is 4:2:0 sub-sampling and (d) is 4:1:1 sub-sampling.

The definitions of the sub-sampled formats [Jack01]:

- **4:4:4 –** there is no chrominance sub-sampling. Each sampled pixel comprises a Y component, U component and a V component. This form of sampling results in the highest digital image quality but requires the largest number of bits to encode.
- **4:2:2 -** there are two chrominance samples of U and V for every four luminance samples. In other words, both of the U and V components are horizontally sub-sampled by a factor of two.
- **4:2:0 -** this name is a bit misleading but what it actually translates to is that for every 2x2 block of Y samples, the U and V components are only sampled once. In effect, both of the U and V components are horizontally and vertically sub-sampled by a factor of two.
- **4:1:1 -** there is only one sample of each of the U and V components for every four horizontal luminance samples, i.e. both of the U and V components are horizontally sub-sampled by a factor of four.

In most commercially available video apparatus, sampled digital pixels generally have values represented by a single byte hence they fall within the range [0, 255]. Although this is most often the case, it is also found within high precision video equipment that digital pixels are represented by an increased number of bits for digitisation accuracy purposes. Sampled pixels represented by 10, 12 or even 14 bits are not uncommon.

In this dissertation, each sampled digital pixel is represented by 8 bits. In the case of a digital greyscale image, each pixel has a range of [0, 255] where 0 represents the colour black and 255 represents white. The values in between represent the different grey levels as you move from black to white. Similarly the U and V components are also represented by 8 bits each.

1.6 The Need for Compression

Digital video is fast becoming the mode of choice as far as the transmission and storage of video is concerned. One of the main reasons for this is that digital video is more versatile than its analogue counterpart. Factors that contribute to its versatility include the ease at which it can be processed by modern day computers and that the process of copying or moving video from one storage medium to another is lossless unlike analogue video whose quality tends to degrade the more it is copied [Done96]. Another factor that is impacting positively in digital video is the continual improvement of digital storage media both in magnitude and processing speed. But if this is the case, why do we bother to compress video?

The answer to this question is due to the voluminous nature of raw digital video. The best way to illustrate this is by means of a mathematical example. Consider a 24-bit RGB image that has a resolution of 512x512 pixels. In raw format, this image would require 786,432 bytes (or 6,291,456 bits). This value actually has little meaning if it is not considered in context with communication systems or networks. If we consider a dial-up Internet connection by means of modem via a telephone exchange, the average available bandwidth is 28,800 bits/second. Therefore to download this image would require 3 minutes and 39 seconds assuming 100% bandwidth utilisation. This length of time is clearly unacceptable in today's fast moving world. Furthermore, if we consider a video clip of similar resolution and with a frame rate of 25 frames/second, it corresponds to a total file size of 196,608,000 bytes. Using the same Internet connection, it would take 15 hours, 10 minutes and 13 seconds to download. If we consider a common Ethernet connection with a bandwidth of 10 Mbits/second instead, to transfer this video clip would require approximately 15 seconds. Even this is extremely demanding in terms of network utilisation. The transmission of such video over RF links where bandwidths are sometimes less than 20 kbits/second is just unthinkable.

It is clearly obvious that in its raw format, digital video like analogue video is not suitable to lower bandwidth applications. However in the digital domain, data is more easily manipulated and as a result of this, a number of compression algorithms have been developed specifically toward video and image compression. For example, if the above image had to be compressed by a factor of twenty, which is easily realisable, instead of requiring 3 minutes and 39 seconds to download via a dial-up modem, it would require only eleven seconds. This duration is obviously far more appealing than the first. There are many different methods of compression, therefore one of the goals of this work is to investigate some of the more common techniques and to implement an application specific video compression system based upon these.

1.7 Contribution of this Work

There are many different techniques that can be utilised in order to compress image and video data both in the spatial and temporal domains. Probably the most commonly used still image compression standard today, JPEG, can be applied to almost any type of image and result in good compression ratios and the preservation of image fidelity [JPEG01]. Without going into too much detail as it is fully described in Chapter 2, JPEG compressed images are divided into square blocks of fixed size and each block is individually compressed. On decompressing the image, the blocks are individually decompressed and placed back into their original positions hence reconstructing the image. It is however possible that a little more compression efficiency could be squeezed out if the system had to be adaptive rather than fixed. In other words, instead of sub-dividing an image into blocks of fixed size and allocating it a certain number of bits, why not divide a low detail piece of that image into a larger sized block and allocate it the same number of bits [Vais92]? This concept applied to the entire image can only improve encoding efficiency. However the complication exists on characterising spatial detail and into what sized blocks pieces of the image should be divided. A significant portion of this dissertation is dedicated to the investigation, implementation and characterisation of an adaptive spatial video frame compression system that is based on the JPEG standard. Where possible, algorithms have been optimised in order to better suit the intended application.

When considering video, existing video compression standards, with the exception of MPEG-4 and H.263, are not geared towards very low bit-rate video applications. MPEG-4 is the first video standard that attempts to solve this problem by separating a video scene into meaningful and identifiable objects [MPEG-4-01]. The separate encoding of these objects rather than treating the video scene as a complete unit, as in the case of the older video compression standards, has opened the door for very low bit-rate video applications. The downfall of MPEG-4 is that is does not specify the actual method by which objects are segmented within a video scene, but rather specifies how to model and encode them once they have been segmented.

There are numerous journal papers that propose different segmentation techniques. It was however found that many of them proved to be too general and as a consequence, the results were average [Kim00], [Smith97] and [Park99]. On the other hand, the more successful techniques tended to require huge amounts of processing using sophisticated systems [Celasun01] and [Han98]. Considering the application specific intention of this investigation, it was therefore decided that a more focussed approach concerning the segmentation and tracking of objects would be adopted. Thus the second major contribution of this work involves the implementation of an object oriented compression system of moderate complexity.

Although two separate investigations are in effect undertaken, the resulting respective systems are ultimately married together to form a single compression system comprising a fully functional encoder and decoder. The performance of the system as a whole is evaluated both objectively and subjectively.

1.8 Thesis Organisation

Although described only briefly, the fundamental concepts of video and compression that were introduced in this chapter serve to prepare the reader for a more in depth discussion in the following chapters.

Chapter 2 provides an overview of existing image and video compression technology. The idea of lossless and lossy compression is described and is then followed by a detailed discussion on some common still image compression techniques employed today. The fundamentals of video compression are then presented followed by an overview of the H.263 compression standard. The chapter concludes by discussing the advantages and disadvantages of several object segmentation and tracking algorithms.

Based on the techniques presented in the previous chapter, Chapter 3 proposes the implemented video compression system and its relevant goals. Design decisions that were made are justified in terms of the systems intended use. Certain enhancements to existing algorithms on which the system is based are proposed. Chapter 4 explains the actual implementation of the system in conjunction with the utilised TriMedia DSP video card. The software implementation of the encoder and decoder is discussed in detail. In some cases, preliminary testing results are presented and encountered problems and their solutions are described. The chapter concludes by mathematically analysing the tracking algorithm and discussing its predicted limitations.

CHAPTER 2

LITERATURE REVIEW

Much literature has been much published on the subject of image and video compression. This chapter summarises the most significant concepts presented in the available literature that is pertinent to this research topic. The various techniques involved in image and video compression as well as object segmentation and tracking are examined.

The review starts by analysing the fundamental techniques that are utilised in still image compression. Most still image compression systems use a hybrid system where image data is initially subject to lossy compression followed by a lossless stage. Such hybrid techniques achieve much higher compression ratios than those techniques that use only one of the techniques. We first take a look at the more commonly implemented lossless techniques followed by an overview of the more popular lossy techniques. The JPEG still image compression standard is examined in greater detail as many of the compression techniques employed within this standard, form the backbone of other video compression standards and techniques.

The concepts and techniques involved in the compression of video will then be presented followed by an overview of the MPEG standards. Following this, the advantages and limitations of currently existing video compression systems will be discussed in context with proposed improved algorithms. Thereafter, the chapter presents an introduction to the concept of video objects and how they can be manipulated in order to maximise compression ratios. The chapter concludes by discussing the criteria for the segmentation and tracking of objects with respect to this work.

2.1 Still Image Compression Techniques

The compression of still images falls into one of three categories: lossless, lossy and a hybrid of lossy and lossless compression techniques. Lossless compression results in an exact replica of the original data source after decompression. Such techniques are generally used in circumstances where it is imperative that the decompressed output is exactly the same as the input, typical applications include the compression of text and medical images. Lossless output compression ratios generally do not exceed a ratio of 2:1 [Bhas00]. Lossy compression techniques on the other hand discard redundant information during the compression stage, hence the decompressed data can never be compared verbatim to its original. With regard to still image compression, spatial data can be discarded due to the limitations of the HVS (as described in Chapter 1), therefore lossy compression results in much higher output compression ratios compared to their lossless counterparts. Clearly, applications that require still images to be highly compressed must therefore be based on lossy compression techniques as lossless techniques alone, would not be able to achieve the required compression ratios.

An improvement with regard to the efficiency of lossy compression systems is that of a hybrid compression system. Still image hybrid compression systems initially compress image data according to lossy means, with the resulting data then being subjected to further lossless compression techniques. Hybrid systems generally result in improved compression ratios compared to systems that employ only lossy compression techniques.

The following sections introduce some of the more pertinent lossless and lossy techniques that exist today. A reasonably detailed description of the JPEG still image compression standard will be presented followed by a discussion of the objectives of a variable block-size transform encoder.

2.1.1 Lossless Compression Techniques

The general concept of lossless compression is the mapping of input data symbols to output code word symbols of shorter length in such a manner that on decoding, the reverse mapping process is applied and an exact replica of the original input data is obtained. Lossless compression algorithms encode data based on their statistical characteristics. The predictability of symbols of a system, termed *entropy* [Shannon48], can be used as a measure of the amount of information conveyed within the data. With regard to images, entropy is used as a lower bound measure of the average number of bits that are required to represent an image [Weeks96]. In other words, the goal of any compression system whether lossy or lossless is to minimise the number of output bits required to represent the original data. The efficiency of an encoder is rated with respect to the input data's calculated entropy as entropy represents the most efficient output bit-rate possible. In information systems, entropy is measured in bits/symbol, however in imaging, entropy is typically measured in bits/pixel.

The entropy of an image is calculated according to (2-1) where p_i is the probability of occurrence of a pixel of value i within the image, G_{MAX} is the maximum pixel value within the image (normally 255) and log_2 is the base 2 logarithmic function. The entropy of a greyscale image is calculated by evaluating the probabilities of occurrence of all pixels within an image and then applying (2-1). The entropy of colour images is calculated somewhat differently. Regarding true-colour images comprising 3 colour planes, i.e. RGB or YUV, the entropy of the image is calculated by the summation of the individual entropy's of each colour plane.

$$Entropy = \sum_{i}^{G_{MAX}} p_i.\log_2\left(\frac{1}{p_i}\right) \tag{2-1}$$

The most commonly known entropy encoder is the *Huffman encoder* [Huffman52]. The Huffman encoder is a lossless compression technique similar to the [Shannon48] encoder in that a complete input data message requires arranging into a table (codebook) of decreasing probabilities. Thereafter, input symbols are mapped to output codewords consisting of a unique sequence of binary bits that vary in length according to each input symbol's probability. The disadvantage of the Huffman algorithm is the possible decoding complexity that could result due to numerous long codewords. [van Voor74] proposed a solution to this problem by mapping input symbols of low occurrence probabilities to output codewords of fixed length while mapping variable-length codewords to the symbols of higher probabilities. This technique ensures that output codewords never exceed a specified number of bits.

An improved version of the Huffman encoder, proposed by Gallager [Gallager78] in 1978 and later improved by Knuth [Knuth85], is the *adaptive Huffman encoder*. Adaptive Huffman encoding maps input symbols to output codewords based on a running estimate of input symbol probabilities. In this manner, the codebook is continually optimised as a function of the dynamic input symbols. A significant advantage of adaptive Huffman encoding is that input symbols do not have to be pre-processed into an occurrence probability table before application of the actual encoding algorithm. In effect, this reduces computation time.

Another common universal entropy encoder is the *Lempel-Ziv* (LZ) or also commonly known as the LZ77 encoder. Initially proposed by Ziv [Ziv77], the LZ77 encoder can be described as a dictionary or look-up based encoder. The encoding concept relies on the assumption that in large blocks of data, patterns such as strings of symbols tend to occur frequently. The storing of frequently occurring symbol patterns in a dictionary allows for each resulting output codeword to consist of a pointer to the position of its corresponding symbol pattern within the dictionary. One of the major limitations of the LZ77 encoding algorithm is the way that it constructs and maintains its dictionary. In 1978, Ziv reviewed the encoding technique and proposed the LZ78 encoder [Ziv78]. More recently, [Welch84] presented an improved encoding algorithm based on the LZ78 algorithm that is better known as the LZW encoder.

However the entropy encoder of interest within this thesis is the *Arithmetic* encoder proposed by Witten [Witt87]. Arithmetic encoding assigns a fractional number of bits per codeword instead of a fixed number of bits as in the case of Huffman encoding. This is achieved by using an interval of real numbers in the range [0, 1) with each input symbol being assigned a range in this interval based on it's occurrence probability. The square bracket in this instance indicates that 0 is included within the range while the closing round bracket indicates that all values up to but not including 1 are included within the range. As the number of input data symbols increases over time, the interval space needed to represent each input symbol becomes smaller resulting in the number of bits required to represent the complete interval growing. Input symbols that have a higher probability reduce the range of the interval space less than symbols of lower probabilities thus adding fewer bits to the output codeword. As the Arithmetic encoding algorithm was chosen to be the lossless mode of compression in this thesis, its detailed operation is best described by an example [Witt87]:

Table 2-1: Set of input symbols with relative probabilities.

Symbol	Probabilities	Sub-divided Input Range
a	0.2	[0, 0.2)
e	0.3	[0.2, 0.5)
i	0.1	[0.5, 0.6)
o	0.2	[0.6, 0.8)
u	0.1	[0.8, 0.9)
!	0.1	[0.9, 1.0)

Consider the probabilities of a set of input symbols applied to an Arithmetic encoder as given in Table 2-1. The starting range of any input message to be Arithmetically encoded falls over the interval [0, 1). The first step of the encoding algorithm is the sub-division of the starting interval into sections that are proportionate to each symbol's occurrence probability. This sub-division is indicated in the last column of Table 2-1.

Consider the transmission of the message: ***eaii!*** as presented by [Witt87]. The encoding process is as follows:

The first symbol to be transmitted determines the new range of the entire interval, i.e. on encoding the symbol *e*, the interval range is scaled from [0, 1) to [0.2, 0.5). The intervals of the remaining symbols are then scaled accordingly.

The previous sub-interval dictates the range of the next sub-interval. As the scale has now changed, the new sub-interval upper and lower limits are calculated according to:

$$.\textbf{lower limit} = \text{Previous}_{\text{low}} + \text{Range}.(\text{sub-interval}_{\text{low}}) \quad (2\text{-}2)$$

$$\textbf{upper limit} = \text{Previous}_{\text{low}} + \text{Range}.(\text{sub-interval}_{\text{high}}) \quad (2\text{-}3)$$

where $\text{Previous}_{\text{low}}$ and $\text{Previous}_{\text{high}}$ are the lower and upper limits of the old interval and

$$\textbf{Range} = \text{Previous}_{\text{high}} - \text{Previous}_{\text{low}}. \quad (2\text{-}4)$$

$\text{Sub-interval}_{\text{low}}$ and $\text{sub-interval}_{\text{high}}$ are the start and ending sub-interval values for the new input symbol as read from Table 2-1, e.g. for the first symbol *e*, $\text{sub-interval}_{\text{low}} = 0.2$ and $\text{sub-interval}_{\text{high}} = 0.5$.

This sub-division process is continued until the range of the last input symbol has been calculated. Figure 2-1 provides a graphic illustration of the encoding process of this example.

The transmitted codeword is chosen as any value that falls within the range of the last input symbol, i.e. [0.23354, 0.2336). In this case, the value to be transmitted is chosen as 0.2335815 as it falls within the final interval range and is easily made up by the binary sequence $2^{-3} + 2^{-4} + 2^{-5} + 2^{-7} + 2^{-8} + 2^{-9} + 2^{-10} + 2^{-13} + 2^{-14}$. The resulting transmitted codeword is a 14-bit string 00111011110011.

From a decoding point of view, the decoder has the original symbol probability table and assumes that the initial interval range was [0, 1). On reception of the binary string, the decoder can automatically deduce that the initial encoded symbol was *e* as the final interval range falls entirely within *e*'s original interval range as given by Table 2-1. Thereafter the decoder simulates the operation of the encoder as it reconstructs a similar sub-division table as given by Figure 2-1. For example, after decoding *e*, the decoder sets it's new interval range to [0.2, 0.5) as was in the case of the encoder. It is then able to deduce that the second encoded symbol was *a* as the new sub-divided interval according to *a* would result in [0.2, 0.26) as once again, the received final interval range falls within this new sub-divided interval. By continuing in this fashion, the original message is decoded.

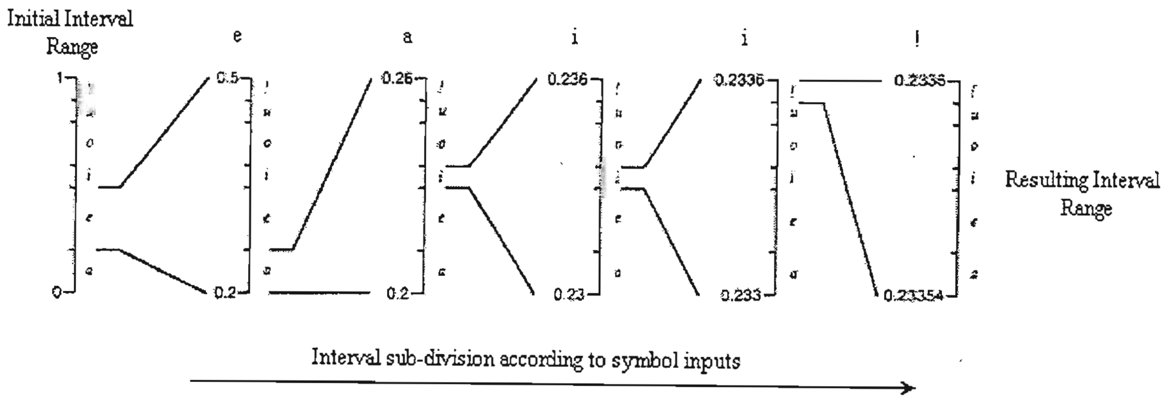


Figure 2-1: Arithmetic encoding example [Witt87].

There are three issues to be considered with Arithmetic encoding. Firstly, the greater the number of input symbols, the more each interval would be subdivided after the encoding of each input symbol hence requiring high precision floating point arithmetic. Secondly, there is no output until all the input symbols have been read. These issues called *interval expansion* and *incremental transmission* were also compensated for in the proposed algorithm by [Witt87]. The last consideration is how the decoder is able to determine when to stop decoding with regard to the received string. Considering the above example, it is possible that the decoder might assume that a string of 0's exist after receiving the 14-bit string. This assumption would naturally result in a decoding error. The simplest method of preventing this error is to use an unique End-Of-File (EOF) symbol [Timmer99]. Thus when the decoder comes across this symbol, it knows that no other data need be considered for the decoding process.

[Howard92] shows that although Arithmetic encoding is computationally more expensive than adaptive Huffman and LZW encoding, it achieves nearly optimal data compression. Results indicate that output compression ratios of an Arithmetic encoding system are generally better than those obtained from its adaptive Huffman and LZW counterparts.

2.1.2 Lossy Compression Techniques

Unlike lossless compression algorithms that on decoding result in an exact replica of the original input data being obtained, lossy compression algorithms discard redundant data during the encoding stage hence the decoded output can never match the original input symbol for symbol. Output compression ratios greater than 30:1 are easily obtained by means of lossy compression compared to the low compression ratios that result with lossless compression. Considering the compression of still images, we have analysed the HVS' insensitivity to high spatial frequencies, especially changes within the chrominance components. Therefore, high spatial frequency image data is considered redundant as its presence within the image is likely to go unnoticed by the average human observer. Thus the goal of any lossy compression algorithm is to take advantage of this redundant information by discarding it. By doing so, high output compression ratios can easily be achieved as there is less image data to encode. However, there is a tradeoff in that the more image information that is discarded, the greater the deterioration of the decoded image.

The distortion that results from a lossy encoded image can be measured both subjectively and objectively. Subjective measurement is the impression that a human forms in his/her mind when viewing a lossy decoded image. It is very difficult to subjectively measure a decoded image according to a predefined scale as different humans will naturally form different impressions of a decoded image. Objective measurement is the process of grading an image according to mathematical formulae. In the case of decoded images, objective measurements generally involve the mathematical computation of the resulting distortion. It will however become evident from the results presented in Chapter 5 that the grading of decoded images or video sequences by objective means alone does not provide an accurate and consistent form of measurement as they tend to be dependent on the contents of the decoded image or video sequence. Thus from a holistic point of view, the most accurate method of grading a decoded image or video sequence would be by both subjective and objective means.

Considering subjective measurement is more of a psychological issue, we expand upon objective measurementation. One of the most common measurement methods is the calculation of the Peak-Signal-to-Noise Ratio (PSNR), in dB's, of the decoded image compared to the original unencoded image:

$$PSNR_{image} = 10.\log\left[\frac{255^2}{\frac{1}{k}\sum_{i=0}^{k-1}(x_i - \hat{x}_i)^2}\right] dB \qquad (2\text{-}5)$$

where k is the total number of pixels within the image, x_i is the i^{th} pixel value within the original image and $\hat{x}_i$ is the i^{th} pixel in the decoded image.

There are three general steps that most hybrid still image compression algorithms follow [Cerveri98]:

1. **Definition of a transform domain:** The approach that a specific compression algorithm employs determines the most suitable mathematical domain in which to operate. The transformation of spatial data into one of these domains is a lossless process.
2. **Quantisation strategy:** This is the process of mapping many input values to a smaller set of output values. This lossy step forms the crux of a lossy compression algorithm as it eliminates redundant image information.
3. **Encoding of the quantised transform coefficients without error:** Once lossy compression has been implemented, higher compression ratios can be achieved by encoding the remaining transformed and quantised image data in a lossless manner.

The lossless compression techniques itemised in step three have already been discussed in Section 2.1.1. The remainder of this section introduces the concept and describes some of the more commonly found transformation techniques as itemised in step one. The transformation technique of interest in this thesis is that of the DCT and will thus be described in greater detail in Section 2.1.3. Finally, quantisation strategies will be discussed in Section 2.1.4 before describing the functionality of a complete still image compression system, namely the JPEG still image compression standard.

The idea of a transform is simply a mathematical tool that is used to convert one set of values to a different set. The different data formats that result after transformation are known as domains and from this point onwards will be called as such. The motivation for transforming a data set from one domain to another is to obtain a more compact representation of the original data set that is more suited to the mathematical operation to be applied to it. Images are generally transformed into one of four domains that include: *spatial domain*, *frequency domain*, *scale-time domain* and the lesser-known *affine transformation*.

Spatial domain encoding is best described by means of an example. Consider the illustration provided in Figure 2-2 in which two areas of varying spatial detail are indicated by white rectangular blocks. The top-left white block illustrates a localised area of low spatial detail as the neighbouring pixels within the defined area have similar values. Another term for the similarity of neighbouring pixels is that of correlation, i.e. pixels that are very similar in value are highly correlated, while dissimilar pixels have a low correlation. Thus instead of sampling each pixel within a highly correlated localised region, compression can be achieved by representing the block of pixels by its average pixel intensity, size and position. The lower block on the other hand isolates an area of high spatial detail as there is a drastic luminance change moving from one region to the other. In order to retain the spatial detail within this particular block, it would prove impractical to represent this area by a reduced set of values that detail only the average information of the localised area. Thus the sampling of definite pixel values within the area ensures that spatial detail is retained at the expense of a lower compression ratio. The difference between compression algorithms that operate in the spatial domain is the manner in which they represent localised regions of an image in terms of mapping to a reduced data set.

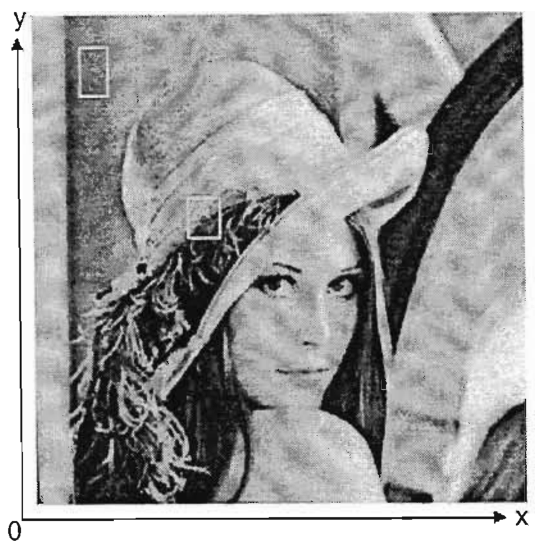


Figure 2-2: Picture of 'Lena' illustrating areas of high and low spatial detail.

The most successful compression systems to date are based on transforms that change image data into the *frequency domain* [Symes98]. The reason for operating in the frequency domain is that it provides a more compact and organised format of the image compared to its representation in the spatial domain. The disadvantage of operating in the frequency domain is the extra computations required to transform the spatial image to the frequency domain for purposes of compression, and then back to the spatial domain for viewing.

Compression is achieved by discarding image information that a human viewer would not be able to detect once it has been transformed back into the spatial domain. One of the more commonly implemented frequency domain compression techniques is that of block-based encoding. Block-based encoding involves sub-dividing the image in the spatial domain into small blocks generally known as *macroblocks*. Thereafter, each macroblock is individually transformed into the frequency domain and encoded. Macroblocks are generally small in size, typically 8x8 pixels in dimension as the bigger the macroblock size, the more computationally expensive the transformation process is.

Examples of frequency transformation algorithms include the Discrete Fourier Transform, Discrete Cosine Transform, Discrete Sine Transform, Discrete Hadamard Transform, and the Karhunen-Loeve Transform (KLT). [Elliot82] conducted tests on the above mentioned transforms for use in image compression applications. Figure 2-3 provides a test image and the respective frequency domain energy compaction plots of the above mentioned transformation techniques. Dark patches within the energy plots indicate areas of low energy content corresponding to redundant image information.

The main objective of any lossy image compression technique is to compact image information in the most efficient manner possible. Based on this objective, [Elliot82] found the KLT to be the most efficient algorithm in terms of energy compaction as from Figure 2-3(f), it is clearly noticeable that the KLT algorithm compacts all it's energy tightly in the top left corner of the output energy plot. The significance of this result is that the compacted energy represents the image data that warrants encoding while the redundant image information represented by the dark area of the energy plot can be discarded. Thus it would be expected that compression by means of the KLT algorithm would result in very high output ratios considering the dark area corresponds to the majority of the energy plot. However the major disadvantage of this KLT algorithm is that it is extremely computationally expensive as it requires the calculation of the eigenvectors and eigenvalues of the autocorrelation matrix of the image hence this transform is not favoured for real-time or fast applications.

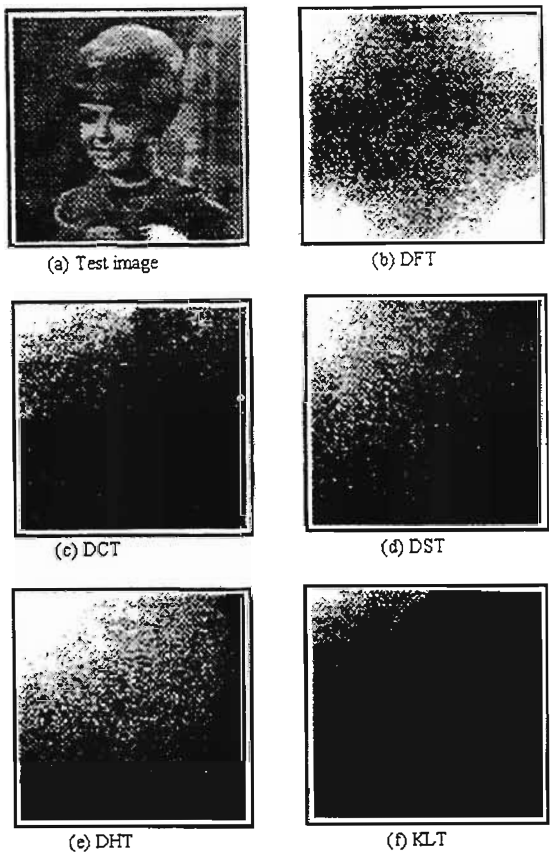


Figure 2-3: Output energy compaction plots of the various transformation techniques [Bhas00].

The DCT [Ahmed74] was found to compare more closely to the optimal characteristics of the KLT than any of the other tested algorithms. From Figure 2-3(c), it is noted that the DCT approaches the energy compaction efficiency of the ideal KLT algorithm. Due to its efficiency and its relative computational simplicity, it is at present the preferred transformation technique for the compression of still images. As the DCT is the transformation technique of interest within this thesis, it is described in greater detail in Section 2.1.3.

Another promising development in the area of lossy compression is that of wavelet transformation [Valens99]. Unlike Fourier based transformations such as the DCT, a wavelet transform has the ability to represent a signal in both the frequency and time domains, better known as the *scale-time domain*. With regard to image processing, the wavelet transform separates a complete image into multiple frequency bands (sub-bands) using both low-pass and high-pass filters. The frequency response of the high and low band-pass filters are defined by a set of basis functions known as wavelets [Polikar99]. By convolving the input image with the defined wavelets, the image is separated into both high and low frequency sub-bands. The rows of the input image are initially filtered and then decimated (sub-sampled) by a factor of two. The columns of the resulting high and low frequency sub-bands are then also filtered and decimated. The result is four sub-bands, each representing different frequency components at a ¼ resolution of the original input image. Figure 2-4 provides an illustration of a typical 2D filter bank and accompanying decimation blocks.

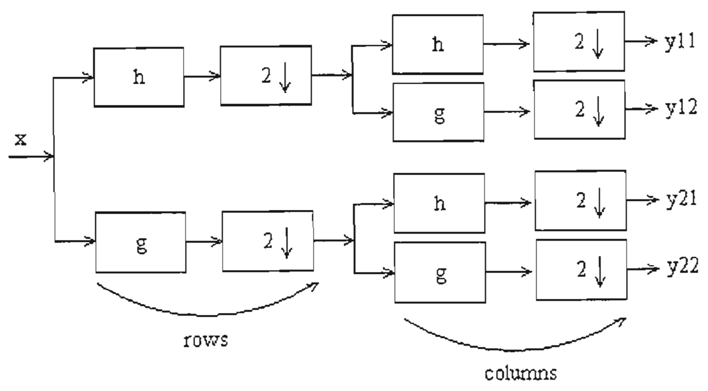


Figure 2-4: Block diagram of a 2D wavelet filter bank [Bhas00].

The 2D input image is represented by x, the low pass filter is represented by h while g represents the high pass filter. After sub-sampling, the sub-band outputs are represented by y_{uv} where typically:

- y_{11} – sub-band representing the average low pass (low horizontal and vertical frequencies) image frequency
- y_{12} – sub-band representing low horizontal and high vertical image frequencies
- y_{21} – sub-band representing high horizontal and low vertical image frequencies
- y_{22} – sub-band representing both high horizontal and vertical (diagonal) image frequencies

Wavelet transformation is an iterative process whereby the low horizontal and vertical frequency sub-band (y_{11}) can be iteratively applied to the filter bank. After each feedback iteration, the horizontal and vertical low frequency sub-band is separated into four further high and low, horizontal and vertical sub-bands thereby improving the energy compaction of the image with each iteration. The number of iterations is a user defined variable. On completion of the last iteration, the final result is a bank of sub-bands decreasing in resolution that on combination, forms the original input image. Lossy compression is achieved by quantising the sub-band values followed by entropy encoding. From a decoding point of view, a similar number of iterations are executed compared to the encoding stage where on each occasion, each sub-band is upsampled by a factor of two and applied to a set of similar band-pass filter banks compared to those used in the encoder.

The popularity of wavelets has increased over the years due to its ability to achieve very high compression ratios whilst still retaining reasonable visual fidelity on decoding. The wavelet transform is being drafted into the proposed JPEG2000 still image compression standard that is still yet to be finalised.

Another technique of compressing and decompressing still images is by means of Fractals. This is a lossy algorithm in which the image is pre-processed and separated into regions of similarity within the spatial domain. The idea behind fractal compression is the automatic detection of a fractal that resembles the image to be compressed. In other words, the function of the fractal algorithm is to find portions of the image that are similar to other portions as determined by an *affine transformation* [Uys00]. The similarity between these portions is defined by a number of translations including rotation, scaling and zooming. The transformation search is highly computationally expensive hence the fractal algorithm is unsuitable for real-time image compression [Komin00]. On the other hand, fractal decompression is much simpler and is thus able to be executed in real-time on an average speed PC.

2.1.3 *Discrete Cosine Transform*

The DCT proposed by [Ahmed74] is currently the most widely utilised transformation algorithm in both still image and video compression applications. The one-dimensional (1D) DCT algorithm is given by:

$$C(m) = \sqrt{\frac{2}{M}}\ .k(m)\sum_{x=0}^{M-1} f(x).\cos\left[\frac{(2x+1)m\pi}{2M}\right] \qquad (2\text{-}6)$$

where **m = 0, 1, ... M-1,** (**M** = total number of array elements to be transformed)

and $k(m) = \frac{1}{\sqrt{2}}$ *for* $m = 0$

$k(m) = 1$ *for* $m \neq 0$

Its popularity is due to the many benefits that it offers [Bhas00]:

- The energy compaction efficiency of the DCT approaches the optimal KLT transform.
- The DCT is orthogonal.
- The 2D DCT is given by:

$$C(m,n) = k_1(m)k_2(n).\sum_{y=0}^{N-1}\sum_{x=0}^{M-1} f(x,y).\cos\left(\frac{m\pi(2x+1)}{2M}\right).\cos\left(\frac{n\pi(2y+1)}{2N}\right) \qquad (2\text{-}7)$$

where

x, m = 0, 1, ... M – 1
y, n = 0, 1, ... N – 1

and

$$k_1(m) = \begin{cases} \sqrt{\frac{1}{M}} & for \quad m = 0 \\ \sqrt{\frac{2}{M}} & otherwise \end{cases}, \qquad k_2(n) = \begin{cases} \sqrt{\frac{1}{N}} & for \quad n = 0 \\ \sqrt{\frac{2}{N}} & otherwise \end{cases}$$

From (2-7) and the orthogonal property of the DCT, the 2D inverse-DCT (IDCT) is given by:

$$f(x,y) = \sum_{y=0}^{N-1}\sum_{x=0}^{M-1} C(m,n).k_1(m).k_2(n).\cos\left(\frac{m\pi(2x+1)}{2M}\right).\cos\left(\frac{n\pi(2y+1)}{2N}\right) \qquad (2\text{-}8)$$

Note the forward 2D DCT and the 2D IDCT are very similar, the only difference is that the coefficients are exchanged in the IDCT. The output of the 2D DCT decomposes an input macroblock into a series of weighted sums of cosine waveforms. With regard to a 2D 8x8 macroblock, Figure 2-5 illustrates the sixty-four cosine waveforms that represent the mathematical coefficients of a transformed macroblock. The top left corner of the transformed output coefficients represents the mean (DC coefficient) of the macroblock in the spatial domain. Moving towards the top right corner and the bottom left corner of the transformed macroblock represents an increase in the horizontal and vertical frequencies respectively (AC coefficients). The bottom right coefficient represents the highest frequency component within the macroblock. Considering that the HVS is unable to detect high frequency changes, lossy compression can be achieved by discarding some of the higher frequency coefficients of a transformed macroblock.

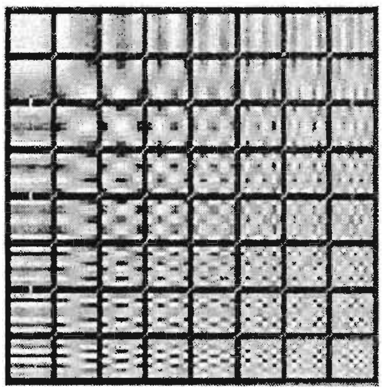


Figure 2-5: Sixty-four representative cosine waveforms of a transformed 2D 8x8 macroblock.

- One of the most important properties of the 2D DCT and the IDCT is that they are separable. From this property, (2-7) can be rewritten in form:

$$C(m,n) = k_2(n).\sum_{y=0}^{7}\left[k_1(m)\sum_{x=0}^{7} f(x,y).\cos\left(\frac{m\pi(2x+1)}{2M}\right)\right].\cos\left(\frac{n\pi(2y+1)}{2N}\right) \qquad (2\text{-}9)$$

The 2D IDCT can be separated in a similar fashion. The implication of this separability property is that instead of operating on a 2D macroblock as a whole unit, a 1D DCT can initially be applied to the rows of the macroblock followed by another 1D DCT applied to the columns of the macroblock resulting in the same transformation of the macroblock into the frequency domain. This separated transformation results in a significant decrease in the overall computational complexity of the applied 2D transform. As a result, many fast 1D DCT algorithms have been proposed that can be adapted for the 2D transformation of images.

Consider the transformation of an 8x8 macroblock by means of (2-7), each of the resulting 64 transformed coefficients would require 64 multiplications and 64 additions assuming the cosine values have been pre-processed. Thus the transformation of an 8x8 macroblock would require a total of 4096 multiplications and 4096 additions. Consider a greyscale image of size 256x256 pixels, this image could be sub-divided into 1024 8x8 macroblocks. The required processing power to transform each macroblock would be considerable. Now consider the row-column approach where a 1D DCT is applied to the rows and then columns of the input 8x8 macroblock. From (2-6), a 1D DCT requires 64 multiplications and 64 additions, thus the application of 16 1D DCT's (8 rows and 8 columns) would require 1024 multiplications and 1024 additions. Thus the separability property of the DCT has already reduced the computational complexity by a factor of four.

The field of fast DCT's has been researched extensively. [Chen77] proposed one of the first fast 1D DCT's based on matrix factorisation and presented it in the form of a signal-flow graph. Several other fast implementations of the DCT have subsequently been proposed. Table 2-2 presents a comparison of the number of multiplications and additions of four proposed fast implementations of the forward DCT. Unless specified, the results indicated are for a 1D DCT and have been obtained from the application of input arrays containing 8, 16 or 32 coefficients. In the case of the 2D DCT algorithms, 2D macroblocks of size 8x8, 16x16 and 32x32 coefficients were applied. The number of multiplications and additions for each IDCT algorithm are the same as that of the forward DCT.

Table 2-2: Results summary of various fast DCT algorithms.

Input co-efficients	**Multiplications**				**Additions**			
	[Chen77]	[Lee84]	[Cho91]	[Fieg92]	[Chen77]	[Lee84]	[Cho91]	[Fieg92]
8	16	12	96(2D)	94(2D)	26	29	466(2D)	454(2D)
16	44	32	512(2D)	-	74	81	2530(2D)	-
32	116	80	2560(2D)	-	194	209	12738(2D)	-

From the above table, the considerable computational savings are noted. What is also noticeable is the increase in the number of required multiplications and additions as the input array size increases. As a result of this property, existing still image and video compression standards that utilise the DCT as their compression agent, generally limit the size of the macroblocks to be transformed to 8x8 pixels. By doing so, this limitation ensures that macroblocks are processed very quickly thus proving suitable for real-time applications.

2.1.4 *Quantisation*

Quantisation is a lossy process that reduces the number of bits required to represent image data. Considering that the transformation of an image, or sub-divided image, into a selected domain is actually a lossless process, compression is achieved by applying a quantiser that irreversibly discards redundant image information.

There are two general types of quantisers: *scalar* and *vector*. The *scalar* quantiser, which is more commonly known, maps a range of input parameters to a single output value [Taub86] such as the staircase quantiser. An increase in the input parameter range results in a coarser quantisation while a decrease results in a finer quantisation. The greater the input parameter range, the greater the compression as more redundant image information is discarded at the expense of a decreasing image fidelity. For more sophisticated applications, a quantiser could be non-linear in the sense that the quantisation range varies according to the type of input, i.e. certain input categories are quantised more coarsely than others.

Vector quantisation (VQ) is the process where two or more image parameters are approximated into one output value. VQ takes an image data subset and attempts to match it with a similar image subset that resides in a codebook [Phamdo00]. In image compression, an image is sub-divided into non-overlapping macroblocks and operated upon individually. The objective of VQ is to match each macroblock within an image to a similar macroblock that is contained within its codebook. Thus instead of encoding the complete macroblock, the vector that describes the position of the best matched macroblock within the codebook is instead encoded thereby achieving compression. The codebook is also transmitted to the decoder for purposes of decoding.

Advanced VQ systems optimise their codebooks according to the diversity of each input macroblock in order to minimise matching errors. Unlike encoding, VQ decoding can be processed very quickly. The decoder uses each macroblock vector and a simple look-up table operation is executed to reconstruct the original image.

2.1.5 JPEG Still Image Compression Standard

A reasonably detailed overview of the JPEG still image compression standard is provided here as it implements many compression techniques that form a basis for those used in existing video compression standards.

JPEG is an acronym for the Joint Photographics Experts Group that was formed when the technical committees of the International Organisation for Standardisation (ISO) and International Telecommunications Union (ITU, then CCITT) joined in 1986. The goal was to establish a joint international standard for the compression of multilevel still images. The result was the publishing of the document entitled "*Information Technology – Digital Compression and Coding of Continuous-Tone Still Images*" or better known as the ISO/IEC 10918-1 standard [JPEG01] in 1992.

The JPEG standard specifies four modes of operation [Penne93]:

- **Sequential DCT-based:** This is the general mode of operation in which an image is encoded from left to right, and top to bottom.
- **Progressive DCT-based:** In this mode, the image is processed in a sequential fashion but encoded by a multiple scanning technique.
- **Lossless:** The image is encoded and decoded in a manner in which no image data is lost. The decoded image is an exact replica of the original digital image.
- **Hierarchical:** In this mode, the image is encoded as a sequence of individually quantised image frames.

The sequential DCT-based format is the most commonly implemented mode of operation thus the remainder of this section is dedicated to a discussion thereof. Considering greyscale images, an input image is sub-divided into 8x8 macroblocks with each macroblock encoded separately. Each macroblock is transformed into the frequency domain with the application of a fast row-column DCT resulting in a 2D macroblock of sixty-four frequency coefficients. Each coefficient is quantised by means of an 8x8 quantisation table by dividing it by its corresponding quantisation coefficient within the quantisation table. Each resultant coefficient is then rounded to its nearest integer value.

Although the JPEG standard does not specify default quantisation tables, it does provide two reference quantisation tables, one table for luminance (Y) component quantisation and the other quantisation table for the two chrominance (U and V) components. The values within the tables were measured from an image with luminance resolution of 720 samples per horizontal line by 576 lines and with a chrominance resolution of 360 samples per horizontal line by 576 lines and with a viewing distance equal to six times the screen width. The original experiment was conducted by [Lohsc84]. The reference quantisation tables are indicated in Table 2-3. From the tables it is noted that the high frequency quantisation coefficients are far greater than the low frequency coefficients thereby resulting in coarser quantisation. Another noticeable feature is that the chrominance quantisation coefficients are greater than those within the luminance table. This is in accordance with the HVS being insensitive to high frequency changes within an image as well as being more sensitive to luminance component changes compared to changes within the chrominance components.

Table 2-3: JPEG reference quantisation tables: (a) luminance quantisation table and (b) chrominance quantisation table.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

(a)

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

(b)

The JPEG standard allows a user to vary the output compression ratio of the encoder by adjusting the coarseness of the coefficients within the quantisation tables. By adjusting a *quality factor* (*Qfactor*), all quantisation coefficients within both the luminance and chrominance tables are scaled accordingly. This adjustment directly affects how harshly each macroblock is quantised thus influencing the output compression ratio.

Following quantisation, each transformed macroblock is entropy encoded. Considering that a majority of the high frequency (bottom right) coefficients within a quantised macroblock would have a value close or equal to zero, it would prove more efficient to encode identified strings of the similar-valued coefficients rather than individual coefficients themselves. A typical method of losslessly encoding strings of similar-valued coefficients is that of *run-length* encoding. The process of run-length encoding is described by an example:

Consider the following consecutive single byte values: 12, 6, 8, 1, 2, 0, 0, 0, 0, 0, 0, 1, 3...

Run-length encoding identifies strings of similar-valued components and represents them in a more compact format, i.e. 12, 6, 8, 1, 2, *, 0, 6, 1, 3...

where '*' is an unique symbol used to identify the start of a run-length encoded section, '0' following the '*' represents the common consecutive value and following '6' represents the number of times the common value appears consecutively. Decoding is achieved by expanding the detected run-length encoded section.

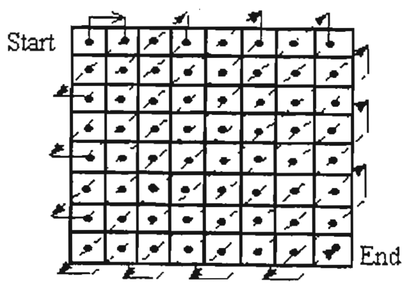


Figure 2-6: Run-length encoding zig-zag scanning pattern.

With regard to the run-length encoding of a quantised macroblock, in order to maximise the number of consecutive similar-valued coefficients (typically but not always 0), the coefficients of the macroblock are scanned and encoded in a zig-zag fashion along the diagonals of the macroblock as indicated in Figure 2-6. By doing so, it ensures that any similar-valued high frequency coefficients are organised in a consecutive order thereby maximising the string lengths of similar-valued coefficients ready for run-length encoding.

The final encoding step is that of entropy encoding by either a Huffman or Arithmetic encoder. The quantised DC coefficient of each macroblock is encoded differently compared to its AC coefficient counterparts. It is assumed that the mean values of neighbouring macroblocks are very similar, therefore instead of encoding the mean of each macroblock as a separate value, in order to improve encoding efficiency, the mean values of neighbouring macroblocks are differentially encoded.

The decoding process is a reversal of the encoding algorithm in that each macroblock is decoded individually and placed in its respective position of the reconstructed image. Figure 2-7 provides the block diagram of a general JPEG encoder and decoder.

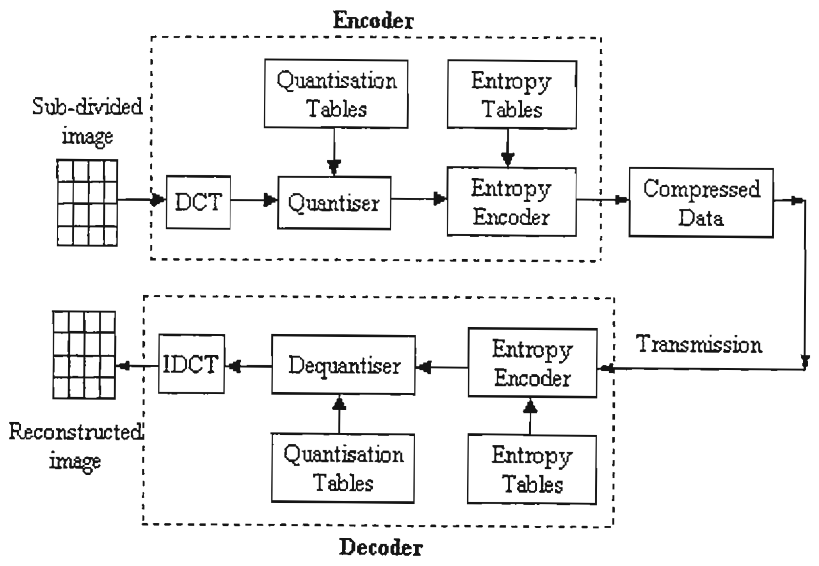


Figure 2-7: Block diagram of JPEG encoder and decoder [Bhas00].

With regard to colour images, the JPEG standard does not set any restrictions on the colour space of an image to be encoded, i.e. an image can be in greyscale, RGB, YUV or any other colour space. Two methods of encoding colour images exist. Either each colour component is operated on individually i.e. the R component is operated on initially followed by the G component and then the B component, or they can be interleaved (mixed during encoding) resulting in slightly better overall compression efficiency [Penne93].

2.1.6 Variable Block-Size Transform

Images that are encoded by the JPEG standard are sub-divided into macroblocks of size 8x8 pixels with each macroblock being encoded separately. This form of fixed block-size sub-division is possibly not the most efficient form of encoding considering an image generally comprises areas of both high and low spatial detail. A more efficient means of encoding would be to sub-divide an image into macroblocks of varying size according to localised spatial detail before encoding. The larger the size of the macroblock to be encoded, the greater its energy compaction as it encloses a larger portion of the image. However a tradeoff exists in that the larger the size of the macroblock, the greater the number of computations required to transform it into the frequency domain by means of the DCT. Due to the development of fast transformation algorithms such as the fast DCT proposed by [Chen77], the transformation of large macroblocks is therefore not of major concern. Thus it would be expected that areas within an image comprising low spatial detail should be sub-divided into large macroblocks in order to maximise energy compaction efficiency. Typical sizes include 16x16 and 32x32 pixels.

On the other hand, a macroblock of size 8x8 pixels or even smaller would be less efficient in terms of energy compaction as it encloses a smaller portion of the image to be encoded. The sub-division of areas of an image comprising high spatial detail into small macroblocks is advantageous in terms of the retention of image fidelity in that within the frequency domain following quantisation, a small macroblock is more likely to contain a larger number of non-zero high frequency coefficients compared to a larger macroblock. From (2-7), this result occurs as the larger the size of the transformed macroblock, the magnitudes of the high frequency coefficients tend to decrease. Thus during quantisation, it is often found that high frequency coefficients of large macroblocks are rounded to zero. Considering that high frequency coefficients represent spatial detail, the greater the number of non-zero high frequency coefficients that remain following quantisation, the greater the retention of image fidelity on decoding.

[Gimlett75] proposed one of the pioneering variable block-size sub-division techniques in which the absolute weighted sum of the transform coefficients provided a measure of the spatial detail of areas within an image to be encoded. Each area is then classified into one of four categories where each category is quantised and encoded differently. This technique results in less bandwidth being allocated to areas with little spatial detail and greater bandwidth allocated to areas of high spatial detail. [Chen77(2)] proposed a similar adaptive encoding technique where an input image, either greyscale or colour, is sub-divided into macroblocks of size 16x16 pixels and quantised according to each macroblocks spatial detail.

[Dins90] proposed an adaptive block-size system that utilises large macroblocks within homogeneous regions of an image, while using small macroblocks near the borders between homogeneous regions. The approach used was bottom-up in the sense that an image is initially sub-divided into 8x8 macroblocks and depending on the AC energy of each macroblock calculated in the spatial domain, neighbouring macroblocks of similar energy levels are clustered together to form larger macroblocks. Having performed the clustering process on the complete image, each resulting variable block-size macroblock is then DCT-encoded and quantised.

[Vais92] proposed a similar variable block-size transform encoder using a top-down quad-tree based portioning approach. The proposed algorithm sub-divides an image into macroblocks of size 32x32 pixels termed *superblocks* with each superblock being encoded separately. The calculated standard deviation is used as an indication of the level of spatial detail of each superblock. After thresholding, each superblock is possibly sub-divided in quad-tree fashion into four 16x16 macroblocks. Thereafter the same procedure is then reapplied to each 16x16 macroblock possibly resulting in each 16x16 macroblock being further sub-divided into four 8x8 macroblocks. The smaller the size of the macroblock, the greater the retained image fidelity on decoding. The proposed algorithm limited the smallest macroblock to a size of 4x4 pixels.

An example of variable block-size quad-tree based portioning is illustrated in Figure 2-8. The resulting sub-divided macroblocks within the original possibly sub-divided superblock are then individually DCT-encoded and quantised. From a decoding point of view, the overhead required to detail the structure of the quad-tree based sub-divided image is restricted by limiting the shape and the number of possible sizes of the final sub-divided superblocks. Results indicated that this form of encoding produced vastly improved output compression ratios for a given PSNR measurement index compared to other encoding techniques.

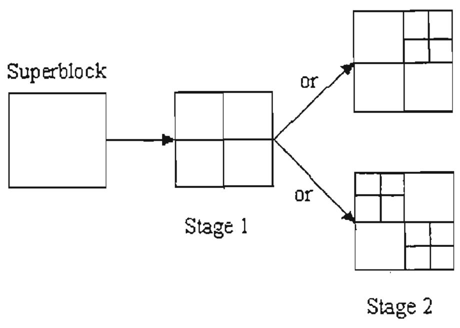


Figure 2-8: Quad-tree based superblock sub-division example.

2.2 Video Compression Techniques

To this point, we have discussed in reasonable detail, some of the more common still image compression techniques. These techniques take advantage of the HVS by manipulating image data on a single frame basis. This form of encoding on a per frame basis is also known as *intraframe* encoding. Considering that the basic form of video is a sequence of consecutive still image frames, a new form of redundancy is introduced known as *temporal* redundancy. Temporal redundancy results due to the high similarity between corresponding areas of consecutive image frames.

A common but simple video compression technique is that of Motion JPEG (M-JPEG). M-JPEG individually compresses each video frame within an input sequence by means of the JPEG still image compression standard. This form of intraframe encoding does not manipulate the temporal redundancy that exists between consecutive video frames resulting in sub-optimal output compression ratios. M-JPEG proved popular before the introduction of temporal based video compression systems. A major disadvantage of M-JPEG is that it is not standardised and thus results in incompatibility problems between the encoding and decoding of compressed video on different computer platforms [JPEG(2)99]. Therefore in order to maximise encoding efficiency, modern video compression systems exploit both spatial and temporal redundancies within the input video stream. Encoding video within the temporal domain is also commonly known as *interframe encoding.*

The following sections provide an overview of the more common lossy video compression techniques. All of the described techniques are essentially extensions of the DCT-based coding techniques implemented in the JPEG still image compression standard.

2.2.1 *Interframe Encoding*

One of the simplest means of manipulating temporal redundancy within a video stream is that of *difference coding* [Manning00]. This process involves the differencing of consecutive video frames resulting in a set of difference frames. These difference frames indicate pixels that have changed in value due to motion within the video scene, hence in order to update areas of motion within a video frame, only the pixels that have changed in value are transmitted to the decoder. The decoder then updates the displayed video frame with the received set of difference pixels. From a lossless point of view, each difference pixel within each difference frame would have to be transmitted to the decoder. Considering that a certain amount of positional overhead would be required for each pixel, this overhead could adversely affect the encoder's output bit-rate. There are two possible solutions to this problem: firstly, introduce a threshold such that only pixels that have changed a significant amount are updated, or secondly, instead of operating on a pixel level, operate on a macroblock level.

Operating on a macroblock level involves the sub-division of each video frame into non-overlapping macroblocks. Thereafter each macroblock within a current video frame could be compared to its previous video frame counterpart in order to detect motion. Only those macroblocks that differ considerably are updated and transmitted to the decoder. The associated positional overhead would be much smaller compared to systems that operate on a pixel basis as pixels are grouped into blocks and only the position of each block requires encoding. The problem with block-based difference coding is that it would prove inefficient in situations where there is a lot of motion. Only macroblocks that remain relatively stationary between consecutive video frames can be efficiently encoded as in high motion scenes, there would be few pixels that would remain unchanged. This problem is alleviated by *motion compensation.*

2.2.2 *Motion Compensation*

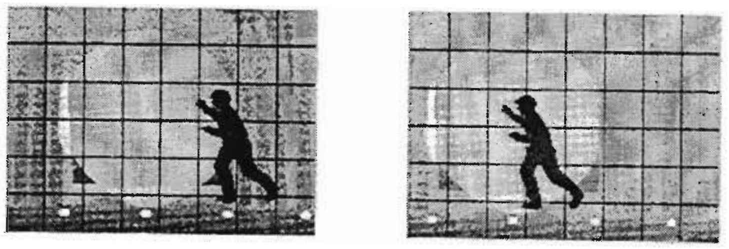


Figure 2-9: Sub-divided consecutive video frames [Symes98].

Motion compensation is the process of reducing the effects of motion within a video sequence by producing an approximation of a video frame based on video data contained in a previous or future video frame [Manning00]. Consider the example of two consecutive sub-divided video frames illustrated in Figure 2-9 where both the object within the scene as well as the video camera are moving. The left frame is the previous video frame while the right frame is the current video frame.

Comparing the two video frames, there are several macroblocks within the same location that would have similar mean values. Instead of encoding both similar corresponding macroblocks, encoding efficiency would be improved by configuring the decoder to use the same block from the previous video frame during the construction of the current video frame. This form of prediction is only effective in areas within a video frame that are not in motion. Several other macroblocks within the video sequence are similar in appearance, however they may not be in the same position within the consecutive video frames. In order to cater for this positional offset, a target macroblock within the previous video frame is selected and a search for a similar macroblock within the current video frame is executed. If a suitable matching macroblock is found within the current video frame, it's positional offset from the target macroblock within the previous video frame is calculated in the form of a (x, y) vector known as a *motion vector*. This searching process is executed for all macroblocks within the sub-divided video frame. If a macroblock is not matched within the current video frame, then it is fully encoded and transmitted to the decoder along with the calculated motion vectors of the matched macroblocks.

From a decoding point of view, each matched macroblock from the previous video frame in association with its positional offset given by its motion vector is used to construct the current video frame. An example of the matching process is illustrated in Figure 2-10. The process of matching similar macroblocks resulting in the calculation of motion vectors is known as *block-based motion estimation.*

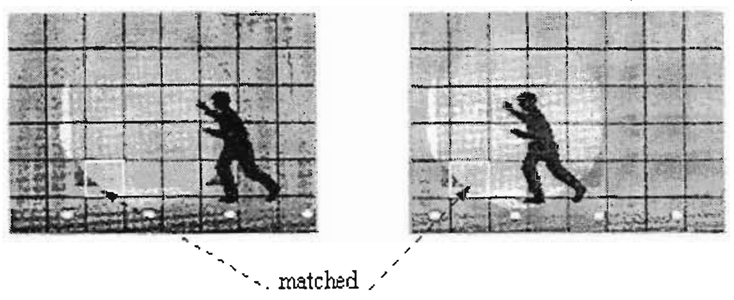


Figure 2-10: Block-based motion estimation [Symes98].

Respective macroblocks between the current and previous video frames are generally matched according to one of two methods: Mean Squared Error (MSE) and Mean Absolute Difference (MAD) [Musman85] where

$$MSE(i,j) = \frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N}\left[s_{k-1}(m,n)-s_k(m+i,n+j)\right]^2 \quad -p \le i,j \le +p \tag{2-10}$$

and

$$MAD(i,j) = \frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N}\left|s_{k-1}(m,n)-s_k(m+i,n+j)\right| \quad -dm \le i,j \le +dm \tag{2-11}$$

Both calculations are performed on macroblocks within the previous video frame (s_{k-1}) and the current video frame (s_k). The matching process is concentrated within a predefined area around the target macroblock termed a *search window*. The search window is generally defined in pixels and in this case is given by (-p, +p) or (-dm, +dm) respectively. A macroblock is best matched when the calculated MSE or MAD values for a particular pixel offset within the search window is minimised. A motion vector for the displacement of the matching block is then calculated. In fast processing applications, the MAD search technique is often chosen before the MSE technique as it does not require any multiplications and is thus executed quicker.

Regarding the matching of macroblocks between previous and current video frames, the most accurate search method would be to calculate the MSE or MAD values at every possible pixel offset within the defined search window. It would be expected that the selection of the smallest MSE or MAD value from the set of results would therefore ensure a best matched situation for a corresponding set of macroblocks. Although this *full-search* technique would result in the most accurate match, if one exists, it is exhaustive and computationally expensive. Several search algorithms exist that find sub-optimal matches while reducing the number of computations required during the matching process. [Jain81] proposed one of the first block-based motion estimation algorithms using a *2D logarithmic* search based on the MSE technique. The algorithm starts by calculating four MSE values at offset pixel positions within a defined search window. The four positions are generally in the shape of a diamond centred around a starting pixel. The lowest MSE value then forms the centre position of the following search iteration but in this case, the pixel distance between the outer search points and the centre of the diamond is decreased. This process continues until eventually only one search point remains. In this case it is assumed that a best match between a macroblock of a previous and current video frame has been found.

[Musman85] describes a *three-step search* that is closely related to the 2D logarithmic search using the MAD criterion. [Sriniv85] outlines both the *conjugate directions* search as well as the *one-at-a-time* search algorithms. Both algorithms prove effective and computationally simple. Newer search algorithms such as the *adaptive search length* [Picker97] have been proposed that achieve similar levels of decoded image quality at only 10% of the calculations required of the full-search algorithm. Another search method is that of the *diamond zonal search* technique proposed by [Tour99]. This method reduces the number of computations required by searching for best-matched macroblocks within a diamond shaped area. Results in some cases indicate that even better visual quality than the optimal full-search algorithm can be obtained.

All search methods described thus far are restricted to searches along integer pixel grids. Such methods search for a best matching macroblock that are offset from each other by discrete pixel steps within a defined search window. This search method is however not as accurate as one that searches for matching macroblocks at a finer resolution, for example half-pixel accuracy [Bhas00]. Half-pixel accuracy search methods interpolate the pixel values between both the target and current search macroblocks by a factor of two. This interpolation allows for the detection of finer motion within a video sequence.

With regard to the encoding of a motion compensated video frame, there are several methods that can be utilised, however we will concentrate on those methods that utilise the DCT. Instead of encoding the complete motion compensated video frame in a similar fashion to that of the JPEG still image compression standard, a more efficient technique is to construct a difference video frame between the motion compensated video frame and the original input video frame. In doing so, it would be expected that the resultant difference video frame would contain numerous zero-valued pixels and would thus improve output bit-rate efficiency. The resulting difference video frame is then encoded similarly to the JPEG still image compression standard.

Figure 2-11(a) provides an illustration of a block-based motion compensated video compression system that utilises an encoding method similar to the JPEG compression standard. The corresponding decoder is illustrated in Figure 2-11(b).

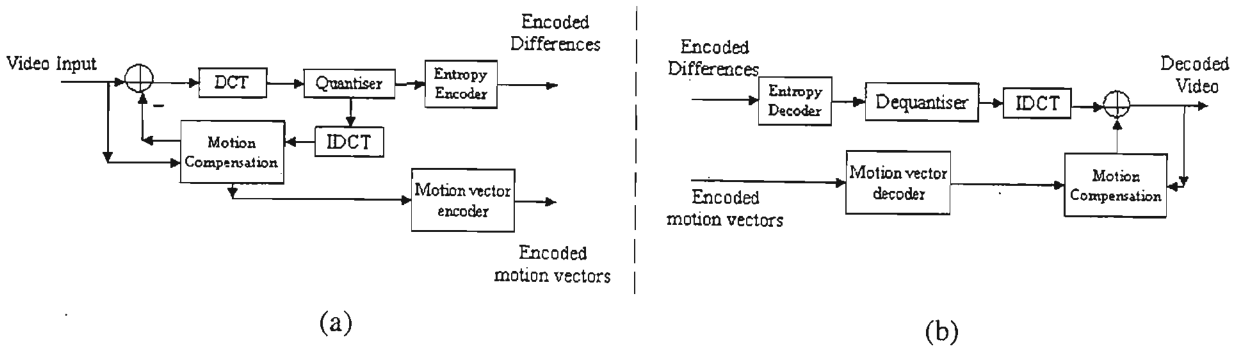


Figure 2-11: Motion compensated video encoder (a) and decoder (b).

2.2.3 *Moving Pictures Expert Group*

In 1988, ISO formed a group known as MPEG in order to develop a common format for the encoding and storing of digital video and associated audio. In 1991, MPEG finalised the first international video compression standard known as the ISO/IEC 11172 *"Coding of moving pictures and associated audio – for digital storage media at up to about 1.5 Mbits/s."* or better known as the MPEG-1 standard [MPEG-01]. Although MPEG caters for the compression of both video and audio, only the encoding of video will be discussed.

Two of the main considerations in the development of the MPEG-1 standard included the need for high compression ratios and the need for random-access capability [Bhas00]. Random-access capability allows the user to access any part of the decoded video stream at any given time, thus intraframe encoding would naturally satisfy this requirement. However, intraframe encoding alone does not cater for the required high output compression ratios, therefore MPEG decided to utilise both intra- and interframe encoding techniques to realise these goals. Within the proposal, it was specified that the encoding of intraframes was to be based upon the JPEG still image compression standard while interframe encoding was to be implemented by means of motion compensation using block-based motion estimation.

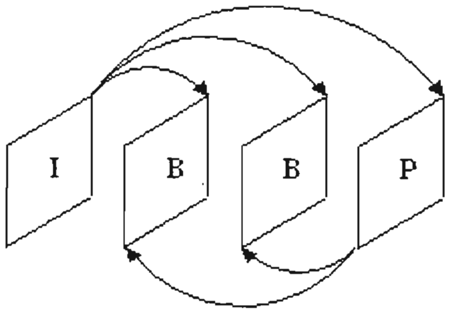


Figure 2-12: Frame types used in MPEG-1 illustrating motion compensation dependencies.

In order to achieve these two goals, MPEG-1 uses three different types of video frames [Wiseman01]:

- The first frame is a reference frame that is encoded independently according to the JPEG still image compression standard. This encoded frame termed an intraframe (I-frame) caters for the random-access criteria of a decoded video stream but at the same time offers only moderate compression. The I-frame is only updated every so often.
- The next type of encoded frame is a predicted frame (P-frame) that is encoded using motion compensation based on previous I- or P-frames. P-frame encoding involves the calculation and encoding of the difference between the current motion compensated P-frame and the previous I- or P-frame. P-frames are compressed much more efficiently that I-frames.
- The last type of frame is a bi-directional frame (B-frame) that is constructed using motion compensated prediction from either previous and/or future I- and P-frames. An example of B-frame prediction is illustrated in Figure 2-12. By interpolating the difference between previous and future frames, the encoding of B-frames result in very high output compression ratios. Considering that B-frames require previous and future I- and/or P-frames to be encoded, the encoded video frame sequence is reordered so that all reference frames required by encoded B- or P-frames

precede the respective frames in the reordered sequence. For example, consider the natural order of a temporally encoded sequence of video frames:

Frame index:	I	B	B	P	B	B	P	B	I
Temporal index:	1	2	3	4	5	6	7	8	9

After reordering the encoded video frames, the transmitted sequence would be as follows:

Frame index:	I	P	B	B	P	B	B	I	B
Temporal index:	1	4	2	3	7	5	6	9	8

MPEG-1 operates on video in the YUV colour format with the luminance and chrominance components sub-sampled according to the ratio 4:2:0. The 3 YUV components of the input video are sub-divided into macroblocks of size 8x8 pixels. The I-frame YUV components are compressed according to the JPEG still image compression standard, i.e. each macroblock is DCT-encoded, quantised and entropy encoded. P- and B-frames differ from I-frames as they actually represent the motion compensated prediction error between the previous and/or future predicted frames. Each resultant error frame is then DCT-encoded, quantised and entropy encoded in a similar fashion to the JPEG standard. MPEG-1 also implements a buffer system that stores compressed video data before transmission to the decoder. Depending on the required output bit-rate and the amount of data within the buffer, a feedback loop from the buffer to the quantiser adjusts the respective Qfactor, which in turn increases, or decreases the output bit-rate.

Once transmitted, the decoder applies an inverse entropy encoder followed by dequantisation and application of the IDCT. Motion compensated P- and B-frames are reconstructed with reference to their respective I-frames. Thereafter the decoded video frames are reordered and displayed.

2.2.4 MPEG-2

The MPEG-2 (ISO/IEC 13818-2) video format, standardised in 1994, was developed for video applications that require high fidelity video at bit-rates between 4 and 100Mbits/s [Vide99]. Similar to MPEG-1, MPEG-2 is a lossy video compression system that is based upon DCT-encoding and block-based motion compensation, as well as predictive and interpolative interframe encoding. The major differences between MPEG-2 and MPEG-1 include the ability to process interlaced video, this falls in line with the original requirements of MPEG-2 namely to transmit broadcast type video; the ability to multiplex video streams, additional colour sub-sampling i.e. 4:2:2 and 4:4:4 sub-sampling is catered for, as well as improved error correction facilities [Vide99]. The most common form of MPEG-2 video has a resolution of 720x480 pixels at 30 frames/second (NTSC). MPEG-2 was designed to be back compatible with MPEG-1.

MPEG-3 was to be the video standard that catered for the needs of HDTV applications. HDTV offers very high quality video and thus requires bit-rates in excess of 80Mbits/s. However, due to the success and versatility of the MPEG-2 standard over a range of bit-rates, development on the MPEG-3 standard was halted as it was discovered the MPEG-2 standard could easily satisfy the requirements of HDTV [Symes98].

2.2.5 MPEG-4

Due to the increasing need for a video compression standard that supports a wide bandwidth range, work on the MPEG-4 (ISO/IEC 14496-2) video compression standard commenced in 1993 with version 2 being finalised in December 1999. The 3 bit-rates that MPEG-4 was initially optimised for include:

- Below 64 kbits/s
- 64 to 384 kbits/s
- 384 kbits/s to 4 Mbits/s

The techniques proposed to achieve such low output bit-rates whilst retaining reasonable video fidelity have revolutionised the way in which video is compressed today. Instead of operating on input video on a frame-by-frame basis and treating each frame as a single entity, MPEG-4 proposed the division of each video frame into meaningful objects. For example, a person walking along a road would be considered an object and the stationary road would be considered another object. The individual objects within a particular video scene rather than the video scene as a single complete entity are then manipulated and encoded individually. This ability to manipulate individual objects within a video scene allows for a new level of interaction between users and visual content [Tosun99].

Three important considerations that are included in the MPEG-4 standard are as follows [Ebrahi00]:

1. **Compression efficiency:** Compression efficiency was the leading principle for the MPEG-1 and MPEG-2 standards. The objective of the MPEG-4 standard was to build on established foundations and improve overall compression efficiency whilst maintaining high decoded video fidelity.
2. **Content-based interactivity:** Segmentation of objects within a video scene enables content-based applications. The representation of objects within a video scene rather than treating a video scene as a whole enables object manipulation, object scalability and bitstream editing. These features are crucial in achieving low bit-rate video compression.
3. **Universal access:** Error resilience was considered a very important factor during the development of MPEG-4. Such resilience allows MPEG-4 to be accessible over a wide range of media, such as mobile networks, portable computers and hand-held communication devices.

The MPEG-4 standard comprises a collection of tools that supports shape coding, motion compensation by means of motion estimation, texture coding, sprite coding and scalability [Ebrah00]. Unlike its predecessors, MPEG-4 separates each video frame into a set of hierarchical layers comprising multiple video objects as illustrated in Figure 2-13.

Layer 1 is the VideoSession (VS) layer that regards an input video frame as a whole entity, similarly to the MPEG-1 and –2 standards. Any 2D object that is segmented within the video scene is termed a VideoObject (VO) comprising the second layer. The shape, motion, texture and identification data of each VO are encoded within the VideoObjectLayer (VOL). Time samples of each VO constitute the VideoObjectPlane (VOP) layer. Grouping of these time samples within the GroupOfVOPs (GOV) layer allows for easy referencing of VOPs in the future.

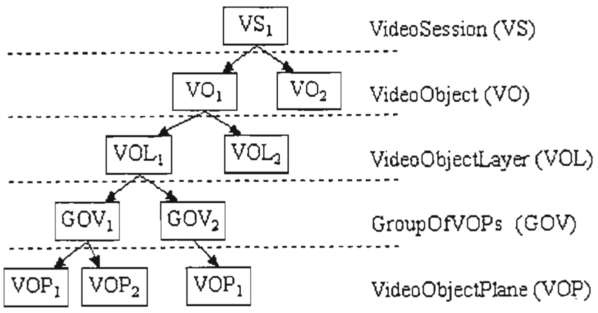


Figure 2-13: MPEG-4 hierarchical layers.

Although the MPEG-4 standard operates on an object level, it does not however specify standard segmentation techniques to be used during the extraction of objects within a video scene. Instead, the standard specifies only the minimum set of functions that are needed to describe a VO once it has been segmented from the rest of the video scene. MPEG-4 specifies 2 types of shape information for segmented VO's, namely binary and greyscale shape information. A segmented object is initially bounded within a rectangular box, comprising non-overlapping macroblocks of size 16x16 pixels. Assume the segmented object is that of a human figure. Naturally a human shape is not rectangular, thus from a shape point of view, only the pixels within the bounded object are wanted while the remaining pixels between the object and bounding block are of no use. Binary shape encoding uses a *binary mask* in which all pixels belonging to the bounded object are set to 1 while remaining pixels are set to 0. This bounded rectangular shape is then encoded block-by-block and transmitted to the decoder. By use of the encoded shape within the binary mask, the decoder is able to determine which pixels of the bounded human figure to display while the remaining pixels are considered transparent and thus have no visual affect on the displayed video frame. The technique of greyscale shape encoding is similar to binary shape encoding, the significant difference being that each pixel within the mask is an 8-bit value. The advantage of greyscale shape encoding is that differing levels of transparency with regard to each object are catered for.

Texture encoding of a sampled VOP is executed in a similar manner to that of the MPEG-2 and MPEG-1 standards with the significant difference being the manner in which macroblocks that fall on VOP boundaries are handled. MPEG-4 sets the pixels that fall between the VOP and the rectangular bounding block to a predetermined value before encoding. The predefined value is chosen in order to optimise the encoding efficiency of each affected macroblock.

MPEG-4 employs motion compensation by means of motion estimation based on the I-, P- and B-frames technique implemented in the MPEG-1 standard. The difference between the techniques is that instead of performing motion compensation within the complete video frame, block-based estimation is concentrated within each bounded VOP thus vastly reducing the overall number of computations required. Motion compensation is executed with half-pixel accuracy. MPEG-4 defines a further type of object called a *sprite*. A sprite is a segmented object or region within a video scene that remains consistent for a period of time. A typical example of a sprite is a background object. Within a static video scene, sprites offer high compression efficiency as they need only be transmitted once to the decoder where they are buffered. Thereafter, a sprite can be cropped or warped and then included within the overall decoded video scene depending on which part of the sprite is visible at any given time.

The last important feature that is defined by the MPEG-4 standard is that of *scalability.* MPEG-4 supports object scalability as well as quality scalability at defined spatial and temporal resolutions [Bhas00]. Depending on defined parameters, spatial scalability determines whether the resolution of a specified VO is increased or decreased, while temporal scalability determines the display rate of each VO. By increasing the display rate of more prominent objects within a video scene, their motion appears more 'normal' and thus improves aesthetic appeal.

At the time of writing, work on the MPEG-7 standard was underway. The intended use of MPEG-7 is somewhat different in comparison to the existing MPEG-1/-2/-4 video compression standards. MPEG-7 is mainly concerned with the efficient labelling of stored data. Data is useless unless it is categorised in terms of what it is, when it was created, who created it etc. A classic example of being unable to locate specific information is executing a search on the Internet. The proposed MPEG-7 standard plans to address this problem by formalising structures of descriptors that fully describe the various forms of multimedia thereby allowing fast and efficient searches whether it be on the internet or a network [Tosun99]. The concept and operation of the proposed MPEG-7 does not fall within the scope of this thesis and will not be discussed further.

2.2.6 H.263

An overview of the H.263 video standard is provided here as it is to be used as a benchmark for the comparison of results obtained in Chapter 5. H.263 is a published ITU video compression standard that is geared towards low bit-rate applications including video-conferencing and video-telephony [H.263]. Such applications generally require bit-rates $\geq$ 64 kbits/s. The H.263 standard is based on many techniques similar to those employed by the MPEG-1 standard as well as its predecessor, the H.261 standard. Due to the differing resolutions and frame rates of the NTSC and PAL video formats, H.263 implements the non-interlaced CIF format of resolution 352x288 pixels.

The structure of the H.263 and MPEG-1 encoder and decoders are similar. Both encoding systems sub-divide the three YUV components of an input video stream into macroblocks of 8x8 pixels, which are then transformed into the frequency domain by applying the DCT. Each macroblock is then quantised and entropy encoded. Motion compensation by means of block-based motion estimation is also commonly implemented in both standards however in the case of the H.263 standard, motion estimation is computed with half-pixel accuracy [Bhas00]. Another major difference between the two standards is the overhead in the signal bit stream is lower for the H.263 standard compared to MPEG-1 thereby making it more suitable for low bit-rate coding.

The H.263 standard allows a system user to configure some advanced encoding options. These include [H.263]:

- **Unrestricted motion vectors:** Beforehand, block-based motion compensation could only be executed within the borders of a video frame. On selecting this mode, H.263 allows motion compensated macroblocks to straddle the frame border, i.e. only a part of a matched macroblock would be displayed. This feature results in a significant compression gain when objects are entering or moving around the frame border.
- **Advanced prediction mode:** In this mode, instead of regarding four 8x8 Y component macroblocks as a single 16x16 block and predicting its future position as in the case of MPEG-1 and H.261 standards, the four 8x8 macroblocks are regarded as individual units and their future

positions are predicted individually. This technique improves the accuracy of the motion compensation stage, however the tradeoff exists in that the greater the number of motion compensated blocks, the greater the overhead required.

- **Syntax-based arithmetic encoding:** Arithmetic encoding is used in this mode resulting in improved entropy encoding efficiency.
- **PB-frames:** A PB-frame consists of two pictures encoded as one unit. The first frame in this unit is a P-frame that is predicted from a previously decoded P-frame, while the B-frame is predicted from the previous and current decoded P-frames.

Although other video compression techniques exist, only the most common techniques have been described here as they provide much of the grounding upon which various other techniques are based. The described techniques thus far, with the exception of MPEG-4, can loosely be described as general video compression systems. General in the sense that they would under most circumstances be able to cope with almost any type of input video stream whether it consists of high or low activity. However the problem with such systems is that they generally require high bit-rates (in excess of 128 kbits/second) in order to cope with the different types of input video. Thus as far as very specific applications requiring low bit-rates are concerned, these general systems would prove inadequate. One of the solutions to this problem that is undergoing intense research and development at present is the concept of object segmentation. Object segmentation is the process of identifying and manipulating individual objects within a video scene rather than operating on the complete video scene as existing video compression standards do today. With this in mind, the purpose of the following section is to describe a few of the more common techniques.

2.3 Object Segmentation

The task of segmenting objects from an input video stream is non-trivial. MPEG-4 is the first video compression standard that achieves low bit-rate video compression by manipulating objects within a video scene. However, the fact that the MPEG-4 standard does not specify the actual means of segmenting video objects is an indication of the complexity and diversity of achieving such a goal. There has been an abundance of research into the field of object segmentation and results indicate that technique efficiencies tend to be application specific. This section provides an overview of some of the more commonly implemented segmentation techniques and discusses the relevant practicalities of each. Once an object has been segmented from a video scene, the question of what do we do with it needs to be asked. The concept and considerations of *object tracking* will be discussed in Section 2.4 before concluding the chapter.

Object segmentation is the process of grouping similar neighbouring pixels into homogeneous regions based on image features. The segmentation of an object can be complex considering the possible presence of shadows, highlights, transparencies and occlusion[5] within a video scene. One of the more commonly implemented methods of segmenting objects within a video scene is by edge detection. Edge detection postulates that a rapid change in neighbouring pixel values exists at the boundary between two regions. [Canny86] developed an edge detection technique that is based on gradients within a localised area to mark the boundary of an object. [Smith97] introduces the SUSAN edge detector as a fast, accurate and noise resistant edge detection method. The SUSAN method utilises non-

[5] Occlusion is the partial or complete obstruction of an object within a scene.

linear filtering to determine which pixels within the image are closely related. In this manner, adjacent pixels that differ considerably are assumed to constitute a boundary. The problem with general edge detection techniques is that they are susceptible to noise[6] resulting in false edge detection. Another disadvantage of edge detection is that often object boundaries are not closed properly. The task of closing the semi-segmented object by means of boundary interpolation is not an easy and accurate one.

Seed growing is another method of segmenting objects. Two common seed growing techniques include region merging and region splitting. Region merging is the process whereby a seed (pixel or defined group of pixels) is placed within an image and the similarity of its neighbouring pixels is evaluated. During the evaluation stage, the differences between the seed pixel and its neighbours are calculated and thresholded. Differences that fall below a defined threshold value are grouped to form regions while differences that exceed the threshold value are assumed to form the boundary between two homogeneous regions.

Region splitting begins with a complete image and divides it into homogeneous regions by means of thresholding. The disadvantage of region based segmentation techniques is that threshold values need to be defined and in some cases, as analysed by Adams and Jones [Adams94], the efficiency of the technique is determined by the selection of the initial seed location. [Park99] describes another form of object segmentation that uses a bottom-up marker extraction approach that sub-divides an image into isolated objects. According to this approach, it is possible that a complete object could comprise several smaller segmented objects, thus post-processing would be required to group similar segmented objects to form a complete object.

2.3.1 *Morphological Segmentation*

Morphology, as described by [Weeks96], is a commonly utilised edge detection method that is used to perform object segmentation. Morphology is concerned with non-linear image processing that deals with the geometrical structure of objects within an image. Traditionally morphology is applied to binary images, i.e. black and white images, but it can however be extended to operate on greyscale images. The following discussion deals with binary morphology only. Two of the more common morphological operators include *dilation* and *erosion*. Dilation is the 'growing' of an object within an image by convolving it with a structuring element. Assuming an object of interest is given by A and a structuring element is given by B, dilation can be defined as all possible vector additions of the elements in A with the elements in B. Letting I^2 define the 2D space of an image, the dilation of object A by structuring element B is given by:

$$A \oplus B = \left\{ t \in I^2 : t = a + b, a \in A, b \in B \right\} \tag{2-12}$$

The structuring element as given by B can be circular, rectangular or any other defined shape. Unlike dilation, erosion decreases the size of an object by convolving it with a structuring element. Erosion is given by:

$$A \ominus B = \left\{ t \in I^2 : t = a + b, a \in A, b \in -B \right\} \tag{2-13}$$

[6] Noise comprises erroneous pixels that occur in either localised or random areas within a scene. An example of noise is randomised black and white pixels known as salt-and-pepper noise.

Figure 2-14 provides an illustration of a typical square structuring element given by B and the result of dilating and eroding an arbitrary object A with B.

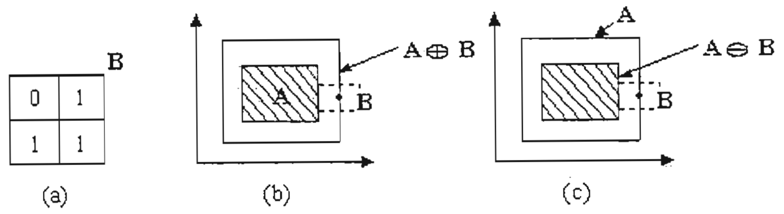


Figure 2-14: Binary morphology: (a) a typical square structuring element, (b) dilation of object A with structuring element B, (c) erosion of object A with structuring element B.

Considering that the dilation of an object results in it growing, by calculating the difference of the dilated object and the original object, the outside contour of the object can be determined. Thus outside edge detection is given by:

$$C_o = (A \oplus B) - A \tag{2-14}$$

The inside contour of an object is determined by calculating the difference of the eroded object and the original object given by:

$$C_i = A - (A \ominus B) \tag{2-15}$$

Full edge detection of an object is realised by combining the results of the calculated outside and inside edges.

Morphology can also be used as an effective noise filter. The application of morphological erosion within an image results in small areas of randomised noise being filtered, while at the same time reducing the size of relevant objects. Thereafter, the application of morphological dilation results in the 're-growing' of the relevant objects. This morphological filtering process is known as *opening* and is given by:

$$A_B = (A \ominus B) \oplus B \tag{2-16}$$

Closing is a further morphological filtering technique that first dilates and then erodes an image. Closing is given by:

$$A^B = (A \oplus B) \ominus B \tag{2-17}$$

2.3.2 *Cluster Segmentation*

Object segmentation can also be achieved by means of clustering. Clustering is the process of grouping similar entities; in the case of images, clustering entails the grouping of similar pixels that belong to a common object. The concept of grouping similar sets of data together to form clusters has been researched in detail by [Everitt93] and [Hart75]. [Everitt93] presents five different cluster analysis techniques to separate groups of similar entities into clusters. These techniques include:

- hierarchical techniques

- optimisation-partitioning techniques
- density or mode-seeking
- clumping techniques
- others

One of the more commonly implemented cluster segmentation techniques is that of *k-means* clustering. This technique randomly allocates a predefined number of cluster centre points, termed *centroids*, within an image [Jain99]. Thereafter, differences between neighbouring pixels are evaluated resulting in pixels being assigned to centroids. Each centroid's position is re-evaluated every time a new pixel is added to its cluster of pixels. This process continues until all the image pixels are assigned to the specified number of clusters. The disadvantage of k-means clustering is that the final number of output clusters must be specified before the algorithm can commence. With regard to video, scenes constantly change therefore it would be virtually impossible to specify the number of relevant clusters that relate to the correct number of objects within the scene. Although techniques have been investigated in order to overcome this disadvantage, segmentation by k-means clustering remains a sub-optimal process [Turi96].

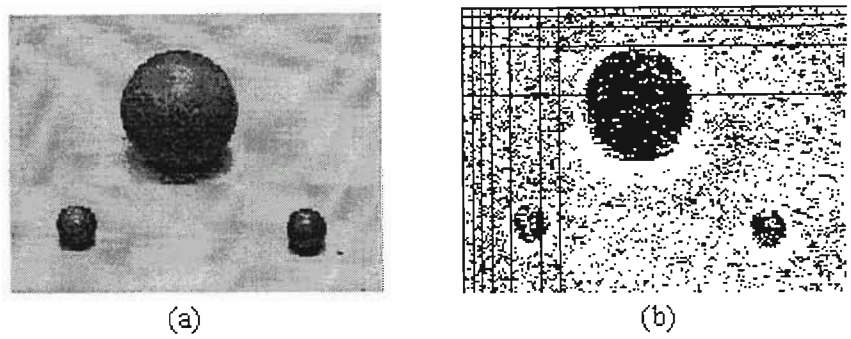


Figure 2-15: Original input image (a), partially sub-divided binary image (b) [Krav99].

[Krav] demonstrated the segmentation effectiveness of the density or mode-seeking clustering technique. The operation of this clustering technique is described by the example provided by [Krav99]: the initial segmentation step was the conversion of an input image, Figure 2-15(a), to binary format by means of a dichromatic reflection model [Krav(2)99]. In this instance, severe noise was introduced as illustrated in Figure 2-15(b). The binary image was then sub-divided into macroblocks of size 8x8 pixels. The mean value of each macroblock was then calculated and mapped to a new image termed a *cluster map*. In this manner, each pixel value within the cluster map corresponds to the mean of a particular binary macroblock. The resulting cluster map is sixty-four times smaller than the original input image as both the number of rows and columns have been reduced by a factor of eight. The original cluster map is illustrated in Figure 2-16(a) on the left side next to an enlarged version. Noise within the cluster map is filtered by means of thresholding. The filtered result is illustrated in Figure 2-16(b).

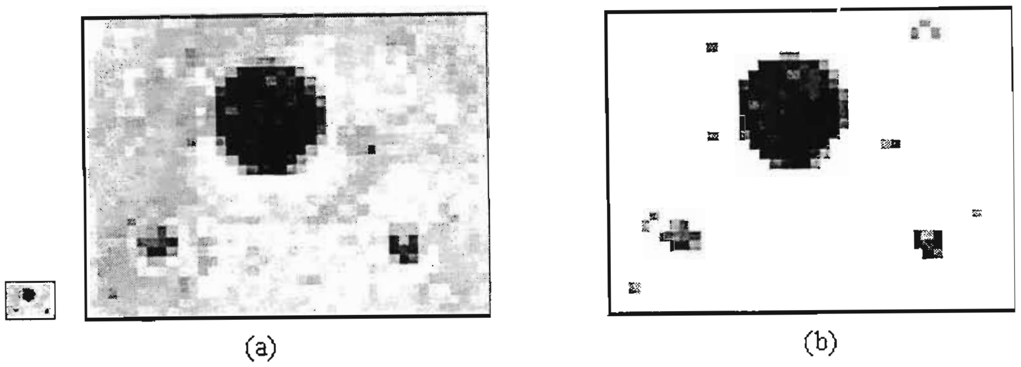


Figure 2-16: (a) Constructed cluster map, (b) filtering by thresholding [Krav99].

The final segmentation step involved the joining of neighbouring cluster pixels into cluster blocks. A defined size threshold is used to determine the minimum allowed size of each cluster block, with those falling below the threshold being filtered. The mapping of the coordinates of the cluster blocks back to the original input image results in the objects within the original image being segmented as illustrated in Figure 2-17. The described clustering technique is advantageous in the sense that the applied mathematical operations are simple and are executed quickly considering a large percentage of the calculations are executed on the cluster map that is sixty-four times smaller than the original input image. The disadvantage of the technique is that the segmented objects are bounded in rectangular blocks.

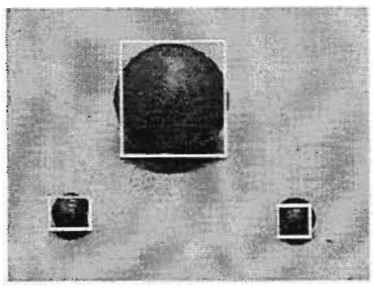


Figure 2-17: Object segmentation by means of cluster block thresholding [Krav99].

2.3.3 *Segmentation Considerations*

There is no one object segmentation technique that proves superior to all others in every situation. The drawback of the discussed segmentation techniques is that they operate on images in the spatial domain only. With regard to video, the goal of object segmentation is the extraction of objects that are in motion. The problem with performing spatial domain object segmentation is the possibility of segmenting objects within the scene that are not in motion. Considering that stationary objects would effectively form part of a background video scene, the segmentation of such objects results in unnecessary computations as well as the possible increase in output bit-rate due to the transmission of erroneous segmented object data.

The first step in segmenting objects within a video sequence is the identification of regions of motion. Once determined, segmentation techniques can be applied within the identified motion regions in an attempt to isolate objects in motion. One of the simplest methods of motion detection is that of frame differencing resulting in the construction of a difference frame. This process requires the calculation of the absolute difference values between corresponding pixels of two consecutive video frames. Large difference values indicate areas of change relating to motion while small difference values indicate areas of little or no motion. [Kim00] implemented an object segmentation algorithm by differencing consecutive video frames followed by the application of the Canny edge dectector. Although this segmentation method is reasonably effective for it's simplicity, post-processing in the form of filtering is required to separate legitimate edges of objects in motion from ambient noise within the difference frame.

[Smith95] introduces a technique that has the ability to segment objects in motion from both a static background as well as a moving background by means of optical flow. Optical flow is concerned with the calculation of motion vectors within the video scene over a set of video frames. The resultant motion vectors of an object in motion are localised and have similar velocity and direction attributes compared to background objects. The grouping of similar localised motion vectors results in the segmentation of objects in motion. Disadvantages of optical flow segmentation include the technique's susceptibility to noise and its inability to handle object occlusion [Marsh94]. Another limitation is the initial rough boundary estimation of a segmented object. The estimated boundary of a segmented object only improves after successive segmentation iterations thus being a function of the number of input video frames. [Han97] implemented a joint motion estimation and segmentation algorithm based on the Markov Random Field (MRF) model for low bit-rate coding. Results proved comparable with the H.263 standard but at a reduced frame rate due to its computational complexity.

A 2D mesh-based framework is another object segmentation technique described by [Celasun01]. This segmentation technique is based on the fusion of node-based motion and triangle-based colour information that requires several input video frames before the actual segmentation process can commence. As well as being computationally expensive, results indicate that this technique is not suitable for real-time video processing. [Gunsel98] detects motion changes within a video sequence by computing the colour content dissimilarity of two consecutive video frames followed by the classification of the results into two classes by means of k-means clustering. The proposed system is application specific and geared towards the segmentation and tracking of human objects by means of applying an adaptive colour thresholding scheme. [Gu98] proposed a hybrid object segmentation system. The postulate is based on the combined activity of both human operator as well as computer. A human operator identifies the objects of interest within the video scene with the relevant boundaries being processed by the computer by means of a watershed transform.

A watershed transform is a morphological operation that separates homogeneous regions from each other. The concept of watersheds is analogous to geological water catchment basins that are divided by watershed lines. A watershed region can be defined as the region over which all points flow 'downhill' to a common point with the actual watersheds separating homogeneous regions from each other. The disadvantage of object segmentation by means of a watershed transform is the tendency to over- or under-segment an image, i.e. an image is divided into too many regions or too few regions. Post-processing would be required to either split over-segmented regions or merge under-segmented regions [Morse98].

2.4 Object Tracking

The segmentation of objects within a video scene is inconsequential unless they are tracked. Object tracking is concerned with the 'following' of an object within the video scene by means of motion modelling and motion prediction. By effectively predicting the future position of segmented objects based on motion models of current and past time samples, low bit-rate video can be achieved as only motion vectors and updated object data need be transmitted to a decoder. An example of object segmentation and tracking is illustrated in Figure 2-18. Figure 2-18(a) illustrates an original video sequence comprising a white ambulance in motion while Figure 2-18(b) provides an illustration of the segmented and tracked object. Instead of segmenting and encoding the white ambulance within each video frame, a more efficient encoding mean would be to segment the object within the first video frame, encode it and transmit it to the decoder where it would be decoded, buffered and displayed. Throughout the following video frames, the encoder tracks the object and transmits updated motion vectors as well as size information to the decoder. The decoder then manipulates the buffered object according to its updated data and then displays it. This technique of tracking segmented objects and transmitting only updated object data allows for very low bit-rate encoding.

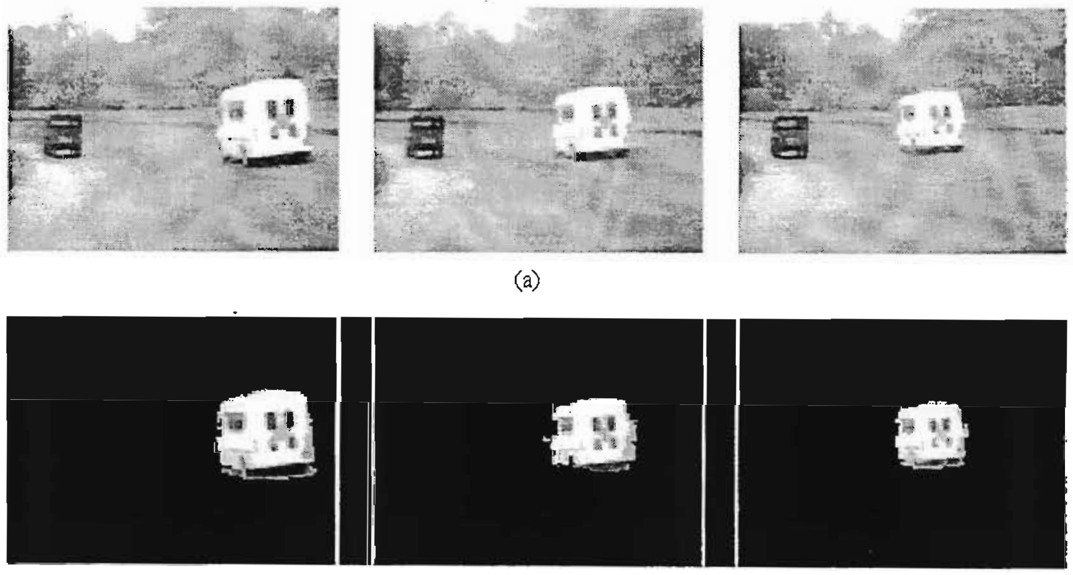


Figure 2-18: Object in motion (a), segmented and tracked object (b).

Many different tracking algorithms have been proposed, some better than others, however no-one algorithm is best suited for all applications. [Brem98] presented a technique for tracking non-rigid (not manufactured) objects in video sequences. The technique assumes prior object segmentation and bounds objects within rectangular blocks. The proposed object tracking model selects the four midpoints along each of the walls of the bounding rectangle as well as the centre of the bounded object and uses these points to track the motion of the segmented object. Future motion of the bounded object is predicted by polygon approximation with each of the five tracking points being tracked separately. The tracking point that best matches the detected moving object defines the object's trajectory. The authors have investigated the technique of region splitting and merging to model the case of partial object occlusion. The disadvantage of this approach is that region splitting and merging requires the definition of thresholds that are not adaptive to the video sequence.

[Meier98] proposed the automatic segmentation of moving objects with respect to both static and moving backgrounds. The technique creates a binary model of the object in motion by means of Canny edge detection. The object is tracked by matching the binary model against subsequent video frames by minimising the Hausdorff[7] distance. The continual updating of the binary model allows for objects to change in shape due to translation, rotation or zooming. Results proved promising but the algorithm was still undergoing further enhancements. [Smith95] described a system in which a segmented object's cluster motion is modelled by an affine model based on optical flow. The relevant motion models as well as the shape of the cluster, represented by a radial map, are used as the comparison attributes during the matching and tracking stages. It is noted that tracking accuracy improves with the increased number of processed video frames. Another significant factor is that the real-time implementation of the system was only possible by means of special hardware.

In image and video processing, an affine model is used to model a 2D object undergoing three-dimensional (3D) motion. A very general, simple form of object tracking is the matching of a segmented object to itself in future video frames. This technique is only accurate if the tracked object undergoes translational motion only. Rotation of an object, or an increase or decrease in size would result in a poor match and even the possibility of loosing track of the object. Affine modelling maps the 3D motion of an object to 2D space therefore improving tracking by catering for 3D variations of motion [Calway00].

[Gunsel98] utilises affine models in the tracking of segmented objects. Each segmented object has its boundary defined by a polygon. Object tracking is performed by selecting points along the polygon boundary and using these points to calculate the displacement vectors between current and future video frames. The change in object shape is compensated for by the warping of the reference object by means of its affine model. The reference object is then updated and combined with the current video frame. Although functional, results proved that the system had difficulty in compensating for the tracking of occluded objects. [McKen99] presents an object tracking system that utilises adaptive Gaussian mixture models. The mixture models describe the composition of a colour object in the HSI colour space. The colour model constantly adapts as the number of tracking iteration increases thereby improving tracking response in differing illumination situations. The system offered reasonable results however the major disadvantage of using colour as the tracking mode is the algorithm's inability to cope with partial or full object occlusion. In such a situation, the Gaussian mixture models become inaccurate resulting in object tracking failure.

Kalman filtering is another commonly used technique for tracking segmented video objects. It is a recursive, predictive update technique that determines the correct parameters of a process model [Welch02]. By effectively modelling the motion of a segmented object and updating the model parameters based on past estimates, Kalman filters can be used to estimate the trajectory of objects in motion. Following each predictive iteration, the model parameters are adjusted with each measurement providing an estimate of the prediction error. Eventually the estimate error measurement converges indicating optimal prediction. An important feature of Kalman filters is their ability to incorporate the effects of noise that arise from both measurements and modelling. They are also frequently used in systems that account for object occlusion due to their ability to estimate future position.

[7] The Hausdorff distance as proposed by [Hutten93] is used as a comparison measurement of binary images. The differences between object edge pixels to be tracked and test points within an image are calculated constituting the Hausdorff distance . The minimisation of this distance results in a best match between the object model and image. This algorithm is computationally efficient, robust to noise and changes in object shape.

Many object tracking systems are limited in their usefulness due to the problem of occlusion. This problem arises when tracked objects pass behind one another or behind stationary objects. In this situation, the tracked object is lost or possibly erroneously merged with another object. Only the re-emergence of the original object would result in the tracking process commencing over again. Occlusion reasoning is the process of compensating for the partial or full disappearance of the tracked object. The art of occlusion reasoning is highly sophisticated and computationally intensive, so much so that an in depth study of this topic would warrant the compilation of a thesis alone. With regard to this thesis, it was decided to concentrate more on the aspects of object segmentation and basic tracking, thus an analysis of occlusion reasoning is not presented.

2.5 Summary

Chapter 2 provides a review of the relevant existing literature on both still image and video compression. The first section of the chapter concentrates on the compression of still images and describes how image data can be compressed by either lossless or lossy means. To date, the DCT has proved the most widely implemented lossy compression technique hence a discussion of its implementation and characteristics was presented. An overview of the JPEG still image compression standard was presented as many of the compression techniques implemented within the standard form the basis of existing video compression standards. The variable block-size transform was introduced as a more efficient alternative to the encoding of a standard 8x8 pixel macroblock.

Thereafter, various techniques involved in the compression of video were introduced. Motion compensation was described as a prediction tool that reduces the redundancy between consecutive video frames in order to optimise output compression ratios. The MPEG video compression standards were also reviewed. The MPEG-4 compression standard introduces the groundbreaking concept of object-based video coding. Instead of encoding a video sequence based on macroblock sub-division as in the case of the MPEG-1 and MPEG-2 standards, MPEG-4 divides a video scene into regions (objects) of varying levels of importance. The application of different prediction and encoding techniques on segmented objects improves output compression ratios. The low bit-rate H.263 video conferencing standard was examined as it is to be used as a comparison measurement for the results obtained from the proposed low bit-rate video compression system in Chapter 5.

The last two sections of the chapter provided an evaluation of objects within a video scene and described how the concept of segmenting (extracting) them from a video scene is non-trivial. Several segmentation techniques are described with the conclusion being drawn that no method proved more efficient in all video applications. The final section described the concept of object tracking and how low bit-rate video is achieved by efficiently tracking a segmented object and transmitting the calculated motion vectors and updated object data to the decoder. Several tracking techniques are described.

The following chapter describes the proposed low bit-rate video compression based on the techniques and concepts that have been reviewed.

CHAPTER 3

PROPOSED VIDEO COMPRESSION SYSTEM

This chapter describes the proposed low bit-rate video compression system that is geared towards the tracking of vehicles in motion. The goal of the thesis is presented followed by a discussion of the building blocks required to implement the system.

3.1 Video Compression System Goal

At the outset of this work, the task presented was to design and implement a complete video compression system comprising an encoder and decoder. The goal of the system was to compress input video suitable for transmission over a low bit-rate channel. Considering the broad nature of the presented topic, it was subsequently refined and decided that in the time available, the two primary objectives of this work should be the implementation of an application specific system that included the ability to compress and decompress video on a frame-by-frame basis, as well as investigate basic object-orientated segmentation and tracking. From an object point of view, it was decided to implement a system that would be geared towards the tracking of vehicles in motion within a low activity video scene. The performance of the implemented system would be evaluated according to output bit-rate of the compressed video, PSNR and general aesthetic appeal of the decompressed video.

Based on the refined topic, the general specifications of the proposed system are as follows:

- Required to transmit streaming compressed video over a low bit-rate channel.
- A basic object-orientated system should be implemented that is geared towards the tracking of vehicles within a generally low activity video scene.
- The system need only cater for the tracking of vehicles moving with translational motion. Rotation and zooming affects of a tracked vehicle need not be considered.
- It can be assumed that the video camera used to provide input video is fixed in a stationary position.
- The transmission channel between encoder and decoder can be assumed error free.
- Decompressed video is to be in true colour.
- Processing and display of video is to be executed in as close to real-time as possible.

3.2 Proposed System Overview

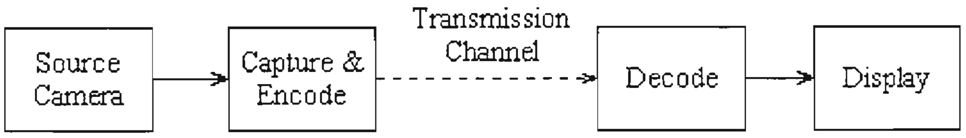


Figure 3-1: Block diagram of complete video compression system.

The proposed low bit-rate video compression system geared towards the tracking of vehicles can be broken into four main blocks as depicted in Figure 3-1. A source video camera fixed in a stationary position is used to provide streaming input video to the system's encoder. Adhering to the proposed system's specifications, it can be assumed that the target video scene comprises low activity, i.e. little motion within the video scene. An example of such a situation is a road within a residential area. It can therefore be assumed that vehicles moving within the target scene would include cars, trucks and motorcycles of varying shape and colour. In order to ensure that vehicles move within the scene with translational motion only, the camera should be positioned in such a manner that the vehicles move generally in an up and down, or sideways direction. An example of a typical target scene is illustrated in Figure 3-2 where it is expected that vehicles would move upwards and downwards along the road within the centre of the video scene.

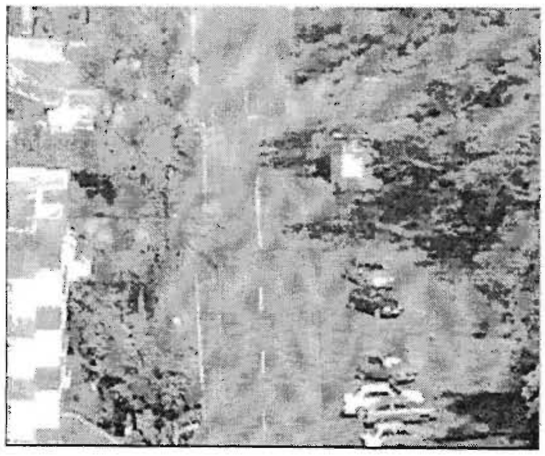


Figure 3-2: Typical target video scene.

Streaming video from the camera is then passed to the encoder. The function of the encoder is twofold. Firstly, the input analogue video from the video camera is digitised and secondly, the digitised video requires processing. Considering the objectives of the proposed system are to encode complete video frames as well as segment and track vehicles within the video scene, it was decided that the encoder should divide a video scene into two main categories and process them separately. These categories include a stationary background scene and objects in motion within the video scene.

The idea of the proposed system is to synthetically create a decoded video scene comprising a background scene and segmented objects. By superimposing segmented objects onto a static background scene, on display, it would appear as if a video scene is natural rather than synthetically created. This concept is illustrated in Figure 3-3. The encoder would encode a selected video frame from the streaming input in which there are no moving objects. This encoded video frame is termed a *reference frame*. The reference frame is to form the background scene of the synthetically created video scene. After encoding, the reference frame is then transmitted to the decoder where it is decoded and buffered within the decoder's memory. Thereafter, the encoder processes the streaming input video frames from the video camera in order to detect objects in motion within the video scene. Detected objects are then segmented, encoded and buffered by the encoder. The encoded object data is then transmitted to the decoder where it is decoded and buffered as well. At any one time, both the encoder and decoder buffer a copy of a segmented object simultaneously.

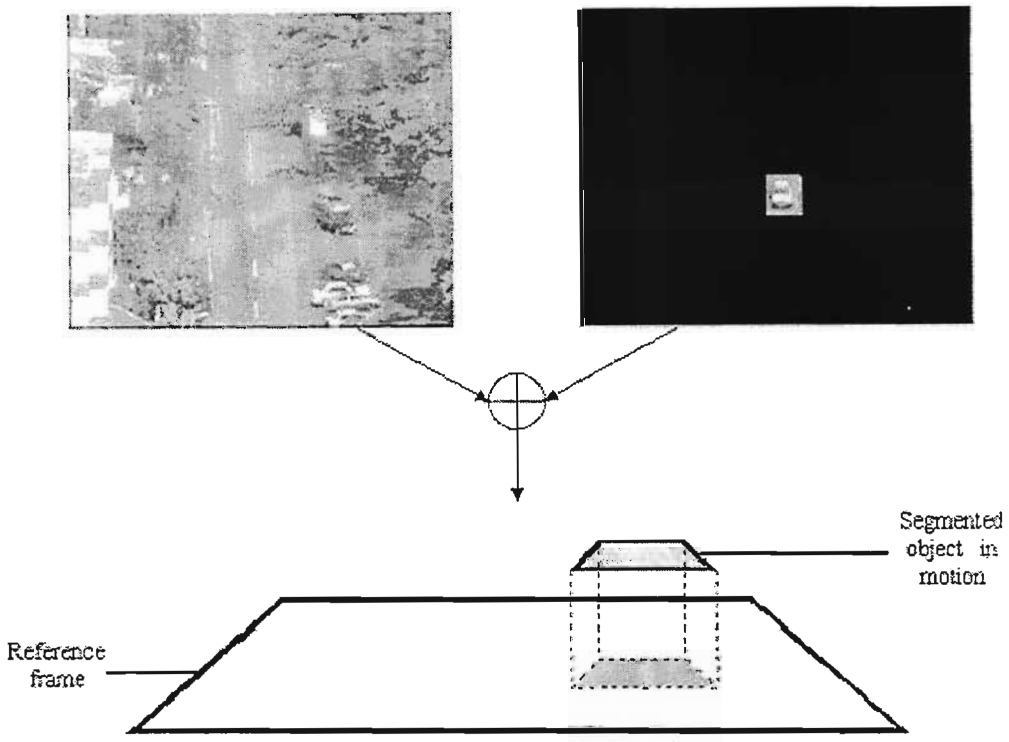


Figure 3-3: Synthetically created video frame comprising a tracked object in motion superimposed onto a reference frame.

Low bit-rate video is achieved by tracking or following the motion of segmented objects within the video scene. During each processing iteration, the proposed system calculates the positional offset, represented by a motion vector, of a segmented object in the current processed video frame with respect to its last recorded position in the previously processed video frame. Each segmented object's motion vector is then transmitted to the decoder. The decoder then re-superimposes each segmented object onto the reference frame according to their updated positions given by their respective motion vectors. The updated video frame is then displayed to the system operator. The continual execution of this tracking and display process would result in an almost 'life-like' decoded video sequence viewed by the system operator.

Most of the required bandwidth of the proposed system would be required by the transmission of an encoded reference frame. However, as the purpose of the reference frame is to provide a background scene on top of which tracked objects are superimposed and then displayed to the system operator, it is not vitally important that the reference frame is updated as often as the tracked objects. Therefore the reference frame need only be updated periodically in order to keep it reasonably current. Following the transmission of a reference frame, the remaining channel bandwidth would be allocated to the transmission of segmented object data and motion vectors of tracked objects.

3.3 Proposed System Design

There are numerous steps involved in the compression of video. Existing image and video compression standards specify the exact algorithms that are utilised in each of the many compression stages. However an abundance of alternate algorithms exist that may outperform those specified in such standards if applied to application specific situations. Algorithmic efficiency and computational complexity are the two key factors that were considered when selecting algorithms to be implemented in the proposed video compression system. This section describes the relevant choices that were made during the design of the proposed video compression system. The actual software implementation of the proposed system is described in greater detail in Chapter 4. As previously described in Section 3.2, the proposed video compression system can be divided into two main parts, the reference frame encoder and decoder and the object segmentation and tracking system. A discussion of design choices pertinent to each of the proposed system's components is presented.

For discussion purposes with respect to the following sections, the proposed low bit-rate video compression system operates on streaming input video that has been digitised. The digital video is in CIF (352x288) resolution and has a frame rate of 25 frames/second. It is in YUV colour space and sub-sampled according to the ratio 4:2:2. The actual video capture and digitisation process is not relevant to this chapter but will however be described in detail in Chapter 4.

3.3.1 Reference Frame Encoder

The design of the reference frame encoder and decoder is based on similar algorithms to those implemented in the JPEG still image compression standard. The reason for this basis is that the JPEG standard has a proven track record of compression efficiency. Their reasonable simplicity and that they are accompanied by an abundance of supporting literature made the algorithms implemented by the JPEG standard more appealing than other researched compression algorithms.

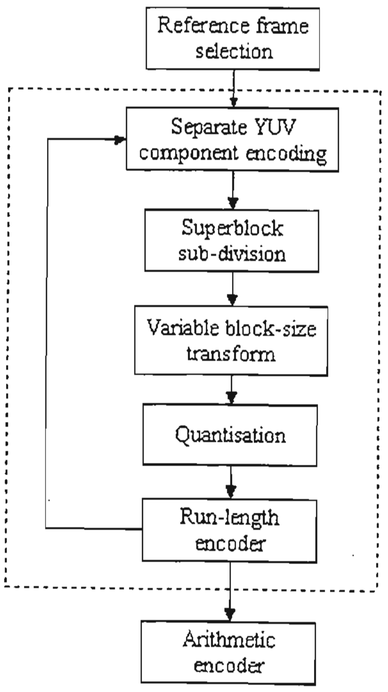

Reference frame selection
Separate YUV component encoding
Superblock sub-division
Variable block-size transform
Quantisation
Run-length encoder
Arithmetic encoder

Figure 3-4: Block diagram of proposed reference frame encoder.

A block diagram of the proposed reference frame encoder is illustrated in Figure 3-4. The first encoding step involves the selection of a reference frame. Although a reference frame has previously been defined as a video frame that contains no motion, it is assumed that the system operator has no prior knowledge of the target video scene, thus any randomly selected video frame can serve as the initial reference frame. It is possible that the selected reference frame may have captured an object in motion, however the periodic yet synchronised updating of a reference frame in the future would ensure that that such selection errors do not occur again.

Considering we are operating the YUV colour space, according to the JPEG still image compression standard, the three YUV components of an image to be compressed are sub-divided into macroblocks of size 8x8 pixels. Each macroblock is transformed into the frequency domain by application of a 2D DCT. The frequency coefficients of each transformed macroblock are then quantised but with the Y component macroblock coefficients being quantised less coarsely than the U and V chrominance component coefficients. The quantised macroblock coefficients are then run-length encoded. Assuming the three colour components of an image are encoded separately, i.e. the Y component is encoded first followed by the U and then V components, the encoded components are finally combined together before Huffman or Arithmetic entropy encoding.

The disadvantage of the JPEG standard is that the algorithm does not differentiate between the encoding of high spatial frequency areas and low spatial frequency areas within an image. Recall from Chapter 2 that a tradeoff exists with the transformation of a defined macroblock into the frequency domain. The larger the size of the macroblock to be transformed, the greater it's energy compaction hence higher encoding efficiency, but at the expense of computational effort. The property of decoded image fidelity as a function of the size of a transformed macroblock was also described. It was discussed that smaller macroblocks have the property of retaining high spatial detail at the expense of encoding efficiency, while large macroblocks result in improved encoding efficiency at the expense of high computational effort coupled with a decrease in retained decoded image fidelity. [Dins90] showed that by adaptively sub-dividing an image into macroblocks of varying size according to measured spatial detail, the overall encoding efficiency was better than that achieved by a system that encoded macroblocks of a fixed size. Based on this finding, it was decided that the proposed video compression system would implement an adaptive algorithm that encoded macroblocks of varying size.

The implemented algorithm is based on an adapted version of the top-down quad-tree variable block-size algorithm proposed by J. Vaisey et al [Vais92]. The goal of the proposed algorithm is to maximise the area of homogeneous regions within the image in order to optimise the resultant encoding efficiency while at the same time retaining reasonable image fidelity. The adapted algorithm operates initially on the luminance Y component followed by the two chrominance components of a selected reference video frame.

Considering that the reference video frame to be encoded is in CIF resolution and is sub-sampled according to the ratio 4:2:2, the proposed algorithm initially sub-divides the Y component into macroblocks of size 32x32 pixels. Continuing with the terminology introduced by [Vais92], macroblocks of size 32x32 pixels from here onwards are termed *Superblocks* as they are the largest sized macroblocks that are operated upon. Thus the sub-divided Y component of the reference frame consists of 99 Superblocks, 11 Superblocks wide and 9 Superblocks deep. Regarding the encoding of the Y component firstly, as in the case of the JPEG compression standard, each Superblock is encoded individually in a left to right, top to bottom direction.

Standard deviation is used as a measure of spatial detail within each Superblock. If the calculated standard deviation of a Superblock exceeds a defined threshold, possibly indicating a high level of spatial detail, then the Superblock is sub-divided in a quad-tree fashion into four 16x16 macroblocks. The partitioning of the Superblock into smaller macroblocks assists fidelity retention of the decoded reference frame. The algorithm continues to evaluate the spatial detail within each sub-divided 16x16 macroblock. The respective standard deviations are calculated and thresholded. Depending on the spatial detail once again, each 16x16 macroblock is possibly sub-divided further into four 8x8 macroblocks. The technique proposed by [Vais92] continues in this fashion with the minimum macroblock size defined at 4x4 pixels. Within the proposed system, it was decided to limit the size of the smallest macroblock to 8x8 pixels. In order to keep track of the structure of each possibly sub-divided Superblock, a header byte is required. The nature and implementation of each Superblock's accompanying header byte is described in detail in Chapter 4.

The U and V reference frame chrominance components are sub-divided in a different manner compared to the luminance Y component. As the HVS is less sensitive to chrominance high frequency changes than luminance high frequency changes, JPEG takes advantage of this fact by quantising the chrominance components more harshly than the luminance components. This effectively results in fewer bits allocated to the encoding of the chrominance components compared to the number of bits allocated to the encoding of the luminance component of an image.

It was decided to adapt this concept and implement a similar system that allocates fewer bits to the encoding of the chrominance components of the reference frame compared to the luminance component. By default, the resolution of the digitised chrominance components of the reference video frame is 176x288 pixels. Considering the Y component of the reference video frame is sub-divided into Superblocks of size 32x32 pixels, ideally the number of sub-divided macroblocks within the chrominance components should correspond to that of the sub-divided luminance component. This would ensure the correct matching of encoded Y component and U and V component macroblocks. If this were the case and considering the resolutions of the Y component and the U and V components, a 32x32 Superblock in the top left corner of the sub-divided Y component would correspond to macroblocks of size 16x32 pixels in both of the U and V chrominance components. Considering the next encoding step is the transformation of each macroblock into the frequency domain by application of a 2D DCT, it was postulated that it would be simpler to implement a 2D DCT algorithm that transformed square macroblocks of varying size rather than transform rectangular macroblocks of varying size. Thus it was decided to reduce the size of the sub-divided U and V chrominance component macroblocks to 16x16 pixels. This is achieved by sub-sampling the chrominance components according to the 4:2:0 ratio. The sub-sampling process involves the omission of every alternate row within both chrominance components resulting in each of the chrominance components having a resolution of 176x144 pixels. Compared to the resolution of the Y component, the two chrominance components are effectively down-sampled by a factor of two in both the horizontal and vertical directions. The matching of a Y component Superblock to its corresponding 16x16 U and V macroblocks is illustrated in Figure 3-5.

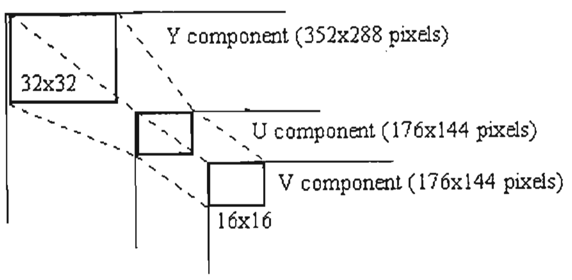

Y component (352x288 pixels)
32x32
U component (176x144 pixels)
V component (176x144 pixels)
16x16

Figure 3-5: Matching of sub-divided Y Superblock to its corresponding U and V macroblocks.

To further increase the overall encoding efficiency of the implemented system, it was decided to reduce the total number of reference frame pixels to encode by not applying the variable block-size quad-tree sub-division algorithm to the 16x16 macroblocks of the U and V components as in the case of the Y component Superblocks. The fixed block size of the two chrominance components ensures minimal bit allocation thereby optimising output bit-rate.

The next encoding step is the transformation of Superblocks and macroblocks into the frequency domain. As in the case of the JPEG standard, the 2D DCT was selected as the means of transforming the sub-divided components into the frequency domain due to its proven energy compaction efficiency and relative computational simplicity. However, a possible implementation disadvantage of the 2D DCT is that as a macroblock size increases, the number of multiplication and addition operations required to transform the macroblock into the frequency domain increases exponentially. The consequences of this disadvantage have since been radically reduced with the development of fast DCT algorithms.

The implemented 2D DCT algorithm is based on that presented by [Chen77]. This algorithm transforms a macroblock into the frequency domain by initially applying a fast 1D DCT to the rows of a macroblock and then to its columns. Although the algorithm presented by [Chen77] is not the quickest in terms of the required number of multiplications and additions compared to the other reviewed algorithms discussed in Chapter 2, it was nonetheless incorporated into the implemented system for two reasons. Firstly, it was one of the few presented algorithms that could be applied to macroblocks of varying size, i.e. 8x8, 16x16 and 32x32 and secondly, it was the only presented journal paper that described that actual mathematical implementation of the fast 1D DCT. [Chen77] described the implementation of their algorithm in the form of a signal flow chart. This flow chart can be viewed in Appendix B.

The sub-divided Y, U and then V macroblock components are transformed into the frequency domain in a left to right, top to bottom order. The frequency coefficients of each transformed macroblock are then quantised. The JPEG standard defines two reference quantisation tables, one for the luminance component and another for the two chrominance components. Each frequency coefficient within a transformed macroblock is divided by its corresponding quantisation coefficient within the respective quantisation table. As the JPEG reference tables are defined for the quantisation of macroblocks of size 8x8 pixels, the coefficients of each table are interpolated in order to cater for the quantisation of 16x16 macroblocks and 32x32 Superblocks. A user-defined quality factor (Qfactor) variable is implemented to allow for the adjustment of the quantisation coefficients in both the luminance and chrominance quantisation tables.

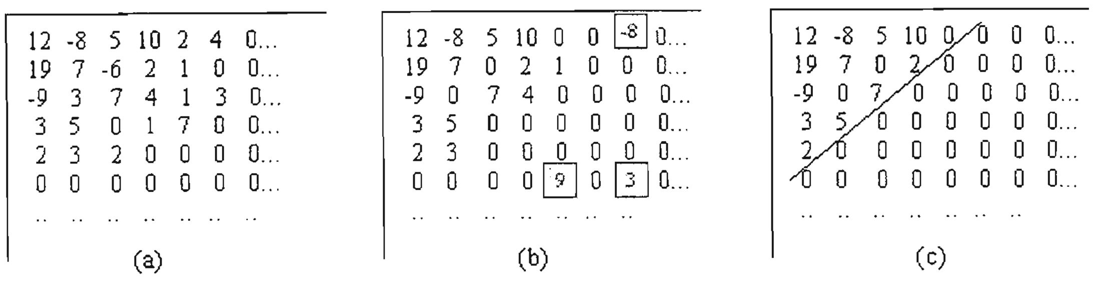


Figure 3-6: Run-length encoding examples: (a) ideal macroblock to be encoded, (b) random non-zero valued high frequency coefficients, (c) proposed algorithm limits remaining low frequency coefficients.

The penultimate encoding step is that of run-length encoding. The quantised coefficients of each Superblock and macroblock are scanned in a zigzag fashion in order to optimise the runs of similar-valued coefficients. It is typically found that 0-valued coefficients are the most common similar valued coefficients that result following quantisation. The efficiency of a run-length encoder is optimal when all of the remaining non-zero valued quantised coefficients are concentrated in the top left corner of the Superblock or macroblock. As a result of this, by scanning the remaining zero-valued coefficients in a zigzag diagonal order, the length of consecutive zero-valued coefficients is maximised thereby resulting in optimal run-length encoding efficiency. This optimal situation is portrayed in Figure 3-6(a). However, the situation often arises where not all high frequency coefficients have non-zero values. An example of this situation is illustrated in Figure 3-6(b). In order to ensure the optimal efficiency of the applied run-length encoder, it was decided to limit the number of non-zero valued coefficients that would be run-length encoded. Considering most of the energy of a transformed macroblock is concentrated within the low frequency coefficients in the top left corner, the idea of the proposed algorithm is to limit the number of remaining non-zero valued low frequency coefficients following quantisation while setting the non-zero valued high frequency coefficients that would detract from encoding efficiency to zero. An example of the proposed low frequency coefficient limitation technique is illustrated in Figure 3-6(c).

The proposed coefficient limiting algorithm superimposes a diagonal line onto the macroblock to be quantised, as illustrated in Figure 3-6(c), and all of the low frequency coefficients to the left of the line are retained while the high frequency coefficients to the right of the line are set to 0. The diagonal line is drawn such that the number of retained coefficients in the top row of the macroblock to the left of the diagonal line is equal to the number of remaining coefficients in the left most column to the left of the diagonal line. In Figure 3-6(c), the number of retained coefficients in the top row and left column is equal to five. Therefore in total, depending on the position of the superimposed diagonal line, the number of remaining low frequency coefficients is limited to 1, 3, 6, 10, 15, 21, 28, 36... It was decided that the proposed algorithm should limit the maximum number of retained low frequency coefficients to 36 irrespective of the size of the macroblock to be encoded, i.e. a maximum of only 36 low frequency coefficients would be retained in 32x32 Superblocks, 16x16 macroblocks as well as 8x8 macroblocks. The tradeoff of the coefficient limitation algorithm is an increase in encoding efficiency as the size of the macroblock to be run-length encoded increases, but at the expense of decreasing decoded fidelity.

The maximum number of retained low frequency coefficients was chosen at 36 in an attempt to ensure the relative simplicity of the implemented limiting algorithm. Considering an 8x8 macroblock, by moving the superimposed limiting diagonal line within the macroblock, the number of retained low frequency coefficients would be limited to 1, 3, 6, 10, 15, 21, 28, 36, 43, 49, 54, 58, 61, 63, 64. On the other hand, depending on the position of the superimposed limiting diagonal line, the number of retained low frequency coefficients within a 16x16 macroblocks as well as a 32x32 Superblocks would include 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78.... From the numbers indicated, it is noted that only the set of numbers in the range [1, 36] are common within all three of the different sized macroblocks. Therefore by limiting the maximum number of retained low frequency coefficients to 36, it ensures that irrespective of the size of the macroblock to be encoded and the position of the superimposed limiting diagonal line with the 36 range, a similar number of low frequency coefficients would always be encoded.

The implemented encoding process is applied to each of the three YUV reference frame components, one at a time. The final encoding step is that of entropy encoding. [Witt87] showed that the encoding efficiency of an Arithmetic encoder is superior compared to the more common Huffman encoder. Based on this finding, the Arithmetic encoder was chosen as the proposed video compression system's entropy encoder. Before entropy encoding commences, the Qfactor as well as the three integer values representing the size, in bytes, of each of the three encoded YUV components are added to the buffer to be entropy encoded. These values are required for decoding purposes. The resulting output file from the Arithmetic encoder is then transferred to the decoder.

3.3.2 Reference Frame Decoder

The reference frame decoding process is simply a reversal of the encoding process. Decoding only commences on reception of the complete encoded file. The received file is initially Arithmetically decoded. The decoded output is in the form of a single file comprising the Y, U and V reference frame encoded components. The dequantisation Qfactor is retrieved from the buffered entropy decoded file as well as the three variables indicating the size of each of the three encoded reference frame components. The decoding algorithm is applied to the Y component firstly followed by the U and V components.

The decoding of the Y reference frame component is more complicated compared to the two chrominance components. According to the Superblock structure defined by it's accompanying header byte, and assuming the Superblock was originally sub-divided into smaller macroblocks, the sub-divided macroblocks are individually run-length decoded and fitted into place reconstructing the original Superblock. The reconstruction process is illustrated in Figure 3-7. The same JPEG reference quantisation tables and interpolated tables that were used during encoder quantisation in relation to the retrieved Qfactor are used to dequantise the respective macroblocks. The dequantised macroblocks within the sub-divided Superblock are then transformed from the frequency domain into the spatial domain by application of a 2D IDCT. The reconstructed and decoded Superblock is then placed into a declared buffer. The decoding process is applied to each of the 99 encoded Superblocks resulting in the Y component of the reference frame being reconstructed in a left to right, top to bottom order.

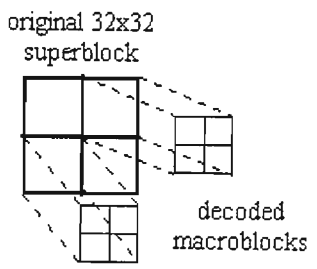


Figure 3-7: Original Superblock reconstructed with it's decoded macroblocks.

Having decoded the Y component of the reference frame, the U and then V chrominance components are then decoded. The encoded data belonging to each 16x16 chrominance macroblock is extracted from the decoded buffer, run-length decoded, dequantised and transformed into the spatial domain by application of a 2D IDCT. For purposes of display, the 4:2:0 sub-sampled chrominance components are required to be upsampled to a ratio of 4:4:4. Two methods of upsampling the chrominance components that were investigated include the polynomial curve fitting technique as well as interpolation by cubic splines technique presented by [Gera94]. However it was decided that the precision and computational complexity offered by both interpolation techniques would not be justified within the intended application. Thus a simpler means of interpolation was implemented. The idea is for a missing pixel to be interpolated from the mean value of its two inline neighbours. This technique is illustrated in Figure 3-8. The figure portrays the proposed interpolation technique applied to the top left chrominance macroblock coefficients. Once the chrominance components have been interpolated, they are buffered accordingly.

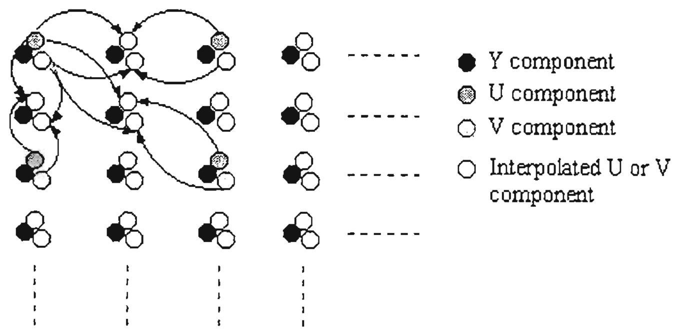


Figure 3-8: U and V component interpolation.

3.3.3 *Object Segmentation*

Having implemented a system that encodes a static video reference frame, the next implementation step of the proposed system is that of object segmentation. Based on the literature review presented in Chapter 2, there are several different approaches to object segmentation. Regarding the project at hand, it was decided to implement a general object segmentation algorithm that would provide reasonable accuracy with moderate levels of computational complexity. An important consideration during the selection of an appropriate segmentation algorithm was that it would be required to operate in conjunction with the implemented reference frame encoder and decoder. From an encoding point of view, an ideal segmentation algorithm would be one that segments objects in the form of rectangular blocks, with the sides of each block preferably being a multiple of 8 pixels. In this case, the existing variable block-size encoder could be used to encode the rectangular objects.

Another factor that required consideration during the segmentation selection process was the ability of the algorithm to segment an object in motion in the presence of ambient noise. The ideal target video scene is one that is completely still except for the occasional moving object. Considering that the video camera was mounted on a tripod during videoing, any slight movement, whether due to wind or an accidental bump, would result in the videoed scene moving. This movement is interpreted as noise within the scene and unless the implemented segmentation algorithm is able to efficiently filter the noise, it is possible that localised areas of noise may be mistaken for moving objects. As was described in the literature review in Chapter 2, the segmentation efficiency of algorithms such as edge detection and optical flow degenerates in the presence of noise. Thus it was decided to implement a segmentation algorithm based on clustering. Cluster segmentation, as presented by [Krav99], enjoys the advantages of computational simplicity while having the ability to filter ambient noise, resulting in reasonable segmentation accuracy. Disadvantages of the algorithm include its dependence on thresholds and its inability to include the sharp edges of a segmented object. The problem of not being able to include the sharp edges of a segmented object is however not an important factor considering one of the requirements of the proposed system is to segment rectangular blunt shaped objects.

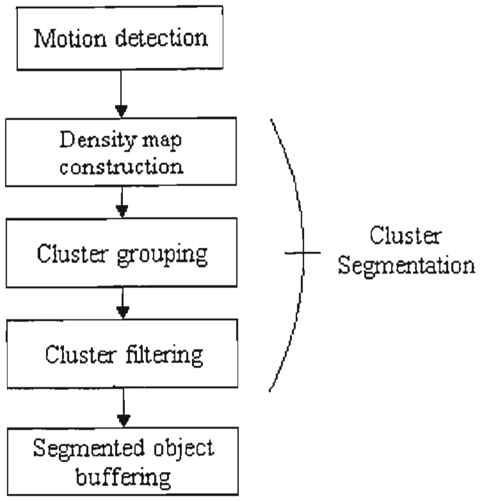


Figure 3-9: Block diagram of proposed object segmentation algorithm.

A block diagram of the proposed object segmentation algorithm is illustrated in Figure 3-9. The first step involved in the segmentation of objects is that of motion detection. Frame differencing is one of the simplest forms of detecting motion within a video scene. It involves the calculation of the absolute difference between corresponding pixels of consecutive video frames resulting in the construction of a *difference map*. The function of the difference map is to distinguish between stationary areas and areas of motion within the video scene. The pixel values of stationary areas throughout consecutive video frames remain fairly consistent therefore the resulting difference pixels are near zero-valued. However, the resulting difference pixels of areas in motion have high values as the respective pixel values differ substantially between video frames. Another characteristic of difference map pixels resulting from objects in motion is that they tend to be concentrated within a localised area.

Traditionally, a difference map is constructed from only the Y component of two consecutive video frames and not the U and V components. However, it was decided not only to construct the Y component difference map of two consecutive video frames, but also to investigate the motion information contained in the U and V component difference maps. The proposed idea is to construct all three YUV difference maps and to verify the consistency of detected localised areas of object motion within each difference map.

A disadvantage of a difference map is that during its construction, slight intensity variations between the input images or minor shifts of the source camera manifest themselves as noise in the output of the differencer and subsequently can easily be mistaken for objects in motion. Thus an important characteristic of an object segmentation algorithm is its ability to distinguish between areas of noise and areas that result due to genuine objects in motion. A key property of noise is that its associated pixels tend to be randomly spaced compared to those pixels of an object in motion. [Krav99] shows that segmentation by clustering is effective in that it is able to separate areas of random noise from genuine objects in motion. Segmentation by clustering involves the sub-division of a difference map into small regions, in the case of the Y component difference map, the sub-divided regions are of size 8x8 pixels while the U and V component difference maps are sub-divided into macroblocks of 4x4 pixels. The mean value of each sub-divided macroblock is calculated and thresholded. The application of a threshold has the affect of filtering macroblocks that contain random noise, as their respective means are much lower than those macroblocks that result due to objects in motion. This is because, as mentioned previously, difference map pixels derived from objects in motion are far more concentrated than those derived from random noise, thus the calculated mean values of objects tend to be higher than that of noise.

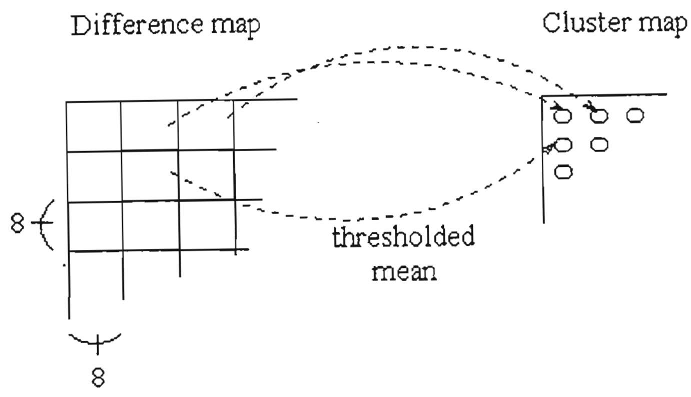


Figure 3-10: Cluster map construction.

The thresholded mean of each macroblock is then mapped to a new image that is termed a *cluster map*. This process is illustrated in Figure 3-10. As a result of the macroblock mean mapping, the constructed cluster map is effectively 64 times smaller than the original difference map. A subsequent advantage of this decrease in size is the reduced effort that is required to process the cluster map compared to the original difference map. The U and V component cluster maps are constructed in a similar fashion.

0	0	0	0	0	0	0	0	0	0
0	0	0	10	8	12	23	0	0	0
0	0	0	34	40	32	29	23	0	0
0	0	0	49	38	42	51	21	0	0
0	0	7	37	28	35	46	41	0	0
0	0	2	12	24	9	23	22	0	0
0	0	0	11	3	10	11	0	0	0
0	0	0	1	4	3	8	1	0	0
0	0	0	0	0	0	0	0	0	0

(a)

0	0	0	0	0	0	0	0	0	0
0	0	10	7	8	0	0	0	0	0
0	0	0	2	4	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	2	12	7	9	23	13	0	0
0	0	0	8	3	10	11	0	0	0
0	0	0	1	4	3	8	0	0	0
0	0	0	0	0	0	0	0	0	0

(b)

Figure 3-11: Grouping cluster elements into cluster blocks, (a) illustrates a typical cluster block resulting from an object in motion and (b) illustrates typical noise cluster blocks.

The next cluster segmentation step is that of grouping neighbouring cluster map elements together. It was decided to implement an algorithm that groups similar elements within a rectangular region together thus keeping in line with the proposed block-based segmentation concept. The function of the algorithm is to group neighbouring non-zero cluster map elements together into rectangular *cluster blocks* as illustrated in Figure 3-11(a). The algorithm applies a further noise filtering step by taking advantage of the discovered property that cluster map elements that result due to objects in motion have much higher values in concentrated areas compared to cluster map elements that result due to noise. Typical examples of noise cluster blocks are illustrated in Figure 3-11(b). Thus it is expected that the calculated mean value of an object cluster block is greater than the calculated mean of a noise cluster block. Therefore by thresholding the calculated cluster block mean values, it would be expected that object cluster blocks would be distinguishable from noise cluster blocks.

The difference between the proposed segmentation algorithm and traditional segmentation algorithms is the use of motion information conveyed in all three YUV components rather than just the Y component. It is postulated that the application of the proposed segmentation algorithm to all of the Y, U and V component cluster maps would result in definite and corresponding rectangular cluster blocks belonging to each object in motion. It is further postulated that rectangular cluster blocks that result due to noise are inconsistent, for example, a region of noise may result in cluster blocks in both the Y and V component cluster maps but not the U component cluster map. Analysis of the consistency of rectangular cluster blocks could possibly allow for further noise filtering. The merits of this proposed filtering concept are discussed in Chapter 4.

3.3.4 Object Tracking

As was described in the overview of the proposed video compression system presented in Section 3.2, low bit-rate video compression can be achieved by means of the proposed encoder tracking segmented objects in motion. Instead of continually re-segmenting, encoding and transmitting the data pertaining to each object, the proposed idea is for the encoder to segment an object once, encode it, and transmit the encoded data to the decoder where it is decoded and buffered. The encoder then tracks the motion of each object and transmits corresponding calculated motion vectors to the decoder. The decoder updates the positional offset of each object according to its received motion vector and then superimposes it onto the reference frame that is then viewed by the system operator.

Considering each object is bounded in a rectangular block constructed from multiples of 8x8 macroblocks, it was decided to implement a type of block-based motion estimation tracking algorithm in order to track an object as it is moving. Instead of matching 8x8 macroblocks between consecutive video frames as in the case of block-based motion estimation within the MPEG standards, it was decided to implement an adapted motion estimation algorithm that would attempt to match the current position of the complete segmented object based on its position within the previous input video frame. A review of the initial proposal however identified several disadvantages:

- The proposed algorithm would not be able to cater for the tracking of objects that increase or decrease in size as they move across the video frame border.
- The tracking of a complete rectangular bounded object would prove extremely computationally expensive if the algorithm has to match objects on a pixel basis.
- The idea of tracking an object by means of block-based motion estimation would be redundant considering the function performed by the original proposed object segmentation algorithm.

The proposed object segmentation algorithm detects motion by means of frame differencing and cluster segmentation. It would not make sense to apply the segmentation algorithm followed by a tracking algorithm that in effect detects the motion of an object as well. An alternate tracking method is to continually apply the proposed object segmentation algorithm and calculate the positional offset of each segmented object within the consecutive input video frames as they are processed. In other words, a segmented object in motion with one video frame would be expected to be segmented once again within the following video frame but in a slightly offset position. Thus the encoder needs only to calculate the positional offset of the object between the two processed video frames in order to keep track of its position within the video scene.

This tracking method would prove advantageous in the sense that the proposed object segmentation algorithm has to be applied continuously anyway in order to detect the presence of new moving objects within the video scene. The problem with this tracking technique is enabling the algorithm to 'understand' that an object that was segmented in one processing iteration is the same object segmented in a displaced position in the following processing iteration. This problem results as the proposed object segmentation algorithm segments objects that are totally independent from those that were segmented during previous segmentation iterations.

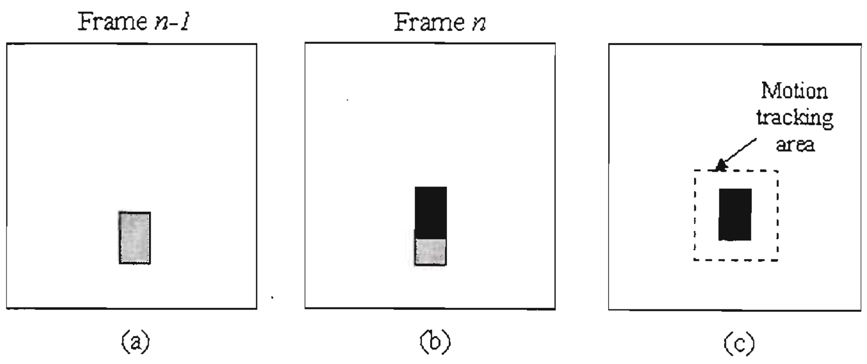


Figure 3-12: Example of proposed tracking algorithm.

The proposed tracking algorithm was revised and it was decided to incorporate the segmentation and tracking concepts into a single complete algorithm. The revised algorithm is best explained by an example. As illustrated in Figure 3-12(a) and (b), a difference map is constructed from two consecutive input video frames. For arguments sake, the two frames are termed Frame *n-1* and *n*. Cluster segmentation is subjected to the resultant difference map in order to segment any objects that are in motion within the video scene with respect to Frame *n*. Figure 3-12(b) illustrates the previous position of the segmented object (grey block) as well as its current segmented position (black block) in Frame *n*. A margin around the location of the current segmented object is then created as illustrated in Figure 3-12(c). The purpose of the margin is to allow a *motion tracking area* for the segmented object. Thus when Frames *n* and *n+1* are differenced and cluster segmented in order to locate objects related to Frame *n+1*, if an object segmented in Frame *n+1* falls within the motion tracking area of an object segmented in Frame *n*, then it is assumed that the object is the same previously segmented object but in a different position. Conversely if the object does not fall within the motion tracking area of the previously segmented object, then it is assumed that the objects are different and are thus tracked independently of each other.

A block diagram of the revised proposed object segmentation and tracking algorithm is presented in Figure 3-13. Each time the algorithm is executed constitutes a processing iteration. The continuous execution of the algorithm ensures that new objects that enter the video scene are segmented and buffered, existing buffered objects are tracked and that objects that no longer exist within the video scene are deleted.

Assuming the proposed segmentation and tracking algorithm is applied to an input video frame for the first time, the YUV components and co-ordinates of each segmented object are extracted from the current input video frame and buffered. Each buffered object is allocated a unique reference number that is required for identification and decoding purposes. A newly buffered object is encoded similarly to a reference frame. However, in order to retain object fidelity, instead of applying a variable block-size encoder to a segmented object, the encoder sub-divides the rectangular object into macroblocks of size 8x8 pixels and encodes each macroblock separately in a left to right, top to bottom order. After encoding the last buffered object, the Y, U and V encoded components of all the buffered objects are finally combined and Arithmetically encoded resulting in a single output encoded object file. This encoded file is then transmitted to the decoder.

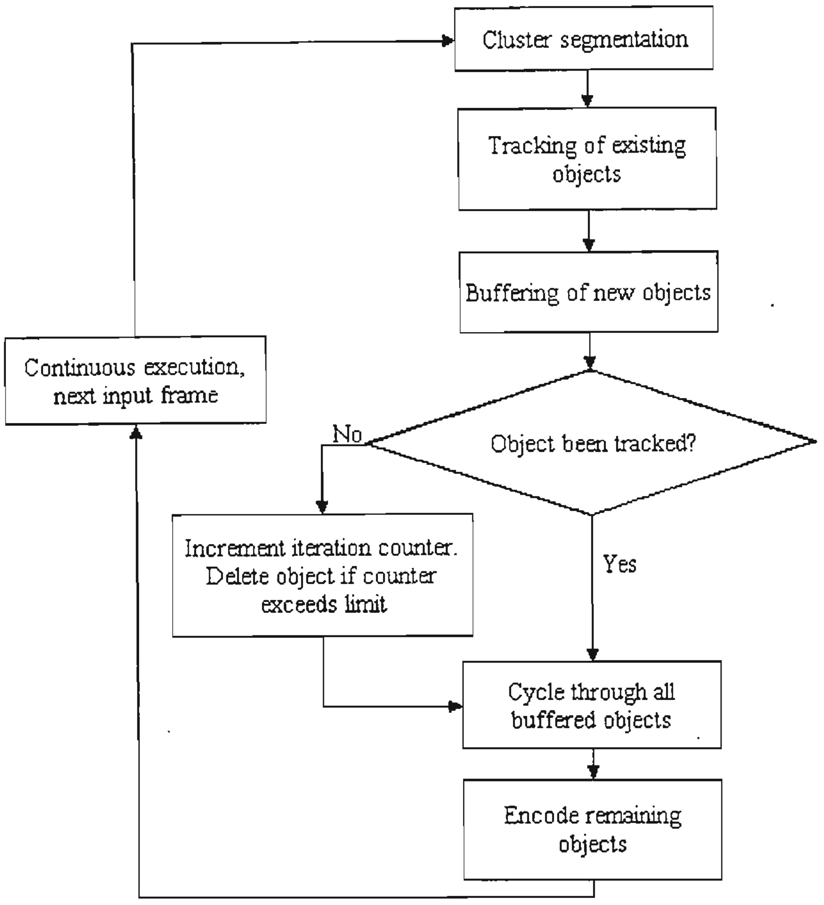

Cluster segmentation
Tracking of existing objects
Buffering of new objects
Continuous execution, next input frame
No
Object been tracked?
Increment iteration counter. Delete object if counter exceeds limit
Yes
Cycle through all buffered objects
Encode remaining objects

Figure 3-13: Proposed object segmentation and tracking block diagram.

Following transmission, the algorithm is then applied once again to the new input video frame. The tracking of existing buffered objects is executed as follows: as per definition of the proposed algorithm, if the co-ordinates of a newly segmented object fall within an existing object's motion tracking area, it is assumed that the two independently segmented objects are actually the same object. The algorithm is extended further to cater for objects that change in size. Changes in size become relevant when objects move across the border of the video scene. For example, a new object 'grows' in size as it enters the video scene while an old object 'shrinks' in size as it exits the video scene. Assuming a buffered object has been successfully matched to its current counterpart, the proposed algorithm compares the sizes of the two objects. If the size difference between the objects exceeds a defined threshold, it is assumed that the object has changed in size due to it moving across the video scene border. Instead of tracking the outdated buffered object, the YUV components of the segmented object within the current video frame are extracted and used to replace the object's previous buffered components. The updated object components are then encoded and ready to be transmitted to the decoder.

On comparing the size of a currently segmented object to its buffered counterpart, if the sizes are similar then it is assumed that the object not moving across the border of the video scene. In this case, the buffered co-ordinates of the segmented object are updated according to the object's position within the current video scene. The motion vector of the current object is calculated and buffered.

Following the tracking stage, it is possible that not all objects that were segmented in the current segmentation iteration correspond to any of the existing buffered objects. This may result if new objects enter the video scene during the current processing iteration. The Y, U and V component data belonging to each newly segmented object is extracted from the current input video frame and encoded. Each new object is allocated a unique reference number.

It is also possible that after the tracking stage, not all existing buffered objects correspond to any of the newly segmented objects. There are four possible reasons why this may occur:

- an error occurred during the current segmentation iteration and an object was not segmented correctly
- the object stopped moving but is still visible within the video scene
- between consecutive processing iterations the object moved out of the video scene
- the object has become partly or fully occluded

When an object moves out of the video scene, it is expected that the tracking algorithm would loose track of it. In this case, the object is deleted in order to ensure that it is not tracked in future processing iterations. Considering that the general operation of the proposed algorithm effectively tracks the motion of an object and not the object itself, if an object had to stop in its path, become partly or fully occluded or be inconsistently segmented between iterations, it is also expected that the algorithm would loose track of the object.

From a processing point of view, it would be incorrect to assume that an object that was not segmented in the current processing iteration but was segmented in previous processing iterations is no longer visible within the current input video frame. In order to cater for this possibility, the proposed algorithm implements an object *iteration counter*. With each processing iteration, every time an existing buffered object is not segmented within the current processing iteration, the buffered object's iteration counter is incremented. At the same time, the object is still superimposed onto the reference frame. If an object's iteration counter exceeds two counts, an object search algorithm is executed. The function of the search algorithm is to locate the buffered object within the current input video frame if it exists. Based on the outcome of the search algorithm, the buffered object will either continue to be tracked in future processing iterations or be will deleted based on the assumption that it is no longer within the video scene.

The proposed search algorithm is an adaptation of the diamond zonal search (DZS) motion estimation algorithm proposed by [Tour99]. [Tour99] proposed a block based motion estimation algorithm that searched within a diamond shaped area for a matching 8x8 macroblock within a current video frame. The proposed algorithm is adapted in the sense that it is not limited to the matching of 8x8 macroblocks, instead it attempts to match the complete Y component of the buffered object to its counterpart in the current input video frame. The algorithm maps an area the size of the buffered object onto the current input video frame at the buffered object's last recorded position. The sum of the absolute differences (SAD) between the Y component of the mapped area within the current video frame and the Y component of the buffered object is calculated. A minimised SAD value indicates a likely match between the buffered object and its counterpart within the current input video frame. The search is initially conducted at the last recorded position of the object, however if a suitable match is not found, then the search is conducted in the surrounding area. The operation of the search algorithm is best explained by means of an example.

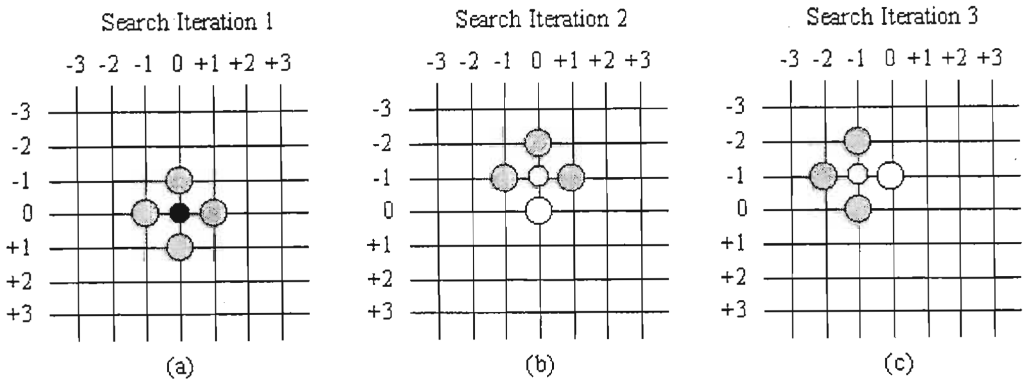


Figure 3-14: Example of diamond zonal object search algorithm.

The first step of the search algorithm maps five areas the size of the buffered object onto the current input video frame. The five areas form the shape of a diamond comprising a centre area surrounded by four areas offset by one pixel above, below, to the left and right of the centre area. An example of the diamond configuration is illustrated in Figure 3-14(a) with the black dot representing the centre area mapped onto the current input video frame. The five SAD values of the bounded areas are calculated with respect to the buffered object. The goal of the algorithm is to place the lowest calculated SAD value in the centre of the search diamond. On comparing the calculated SAD values, if the centre value is not the lowest, the diamond moves in the direction of the lowest SAD value. On moving, the SAD values of the new search points within the diamond shape are evaluated and compared once again. The comparison and moving process is executed over until eventually the lowest SAD value falls within the centre of the diamond search area. When this situation occurs, it is assumed that a best match between the buffered object and the 'object' within the current input video frame has been found. The search is executed within a limited area in order to avoid the situation where it is executed endlessly because no suitable match between the buffered object and that within the current input video frame is obtained. The defined search area of the example illustrated in Figure 3-14 is limited to ±3 pixels in both the horizontal and vertical directions with respect to the original starting search area.

With regard to the example, Figure 3-14(b) illustrates the diamond shape moving in the direction of the lowest calculated SAD value, which in this case is the search area above the initial starting search area. Considering the SAD values of the new search areas at co-ordinates (0, -1) and (0, 0) have already been calculated in the first search iteration, they are not re-evaluated in the second search iteration. The three SAD values of the new search areas (grey dots) within the diamond shape are calculated and compared to each other once again. In this case, it is found the search point to the left of the centre of the diamond has the lowest SAD value thus the diamond shape moves one pixel to the left as portrayed in Figure 3-14(c). The three new search areas (grey dots) are mapped and the respective SAD values are calculated and compared once again. In this iteration, it is found that the centre search area results in the lowest SAD value thus the search algorithm is terminated as it is assumed that the buffered object has been suitably matched within the current input video frame.

As a final test to ensure that the matched area within the current input video frame is in fact the correct counterpart of the buffered object, the mean values of the Y components of the two objects are calculated and compared. If they fall within a specified range, then it assumed that a best match has been found. Recall, the purpose of the search algorithm is to determine whether or not a previously tracked object is still visible within the current input video frame. In the above example from the results, it is assumed that the buffered object is still visible within the current video frame. Thus the vertical and horizontal offset of the diamond's centre search area is calculated (in this case (-1, -1)) with respect to the original starting search location. This calculated offset is used as the motion vector that would be transmitted to the decoder. The object's iteration counter is reset ready for the next processing iteration.

In the situation that a buffered object is not matched to its counterpart within the current input video frame, it is then assumed that the buffered object is partly or fully occluded or it is no longer present in the current video frame. In this case, the buffered object is then deleted. The object is not tracked within the following processing iteration. The search algorithm is applied to all objects whose iteration counters exceed two counts.

The final step of the proposed segmentation and tracking algorithm is that of encoding. The YUV components of newly segmented objects as well as those objects whose YUV components have been updated due to the object changing shape, are fully encoded, i.e. the 8x8 macroblocks constituting the complete object are DCT transformed, quantised and run-length encoded. The encoded YUV components of the objects along with the motion vectors and reference numbers of existing buffered objects are finally combined and Arithmetically encoded. The encoded file is then transmitted to the decoder.

Although not a specified objective, an advantage of the proposed search algorithm is its ability to tolerate occlusion. In the event that a search algorithm is executed, if it is able to detect even a small part of an occluded object, then the tracking of the object continues as per normal. If however the search algorithm determines that the object is too occluded for satisfactory tracking, the buffered object is then deleted. In the event of the object reappearing, the proposed segmentation algorithm would detect the 'new' object, segment, encode it and track from there onwards. The drawback of re-encoding an emerged object is it's negative affect on output bit-rate as each time an object is fully encoded, it requires far more output bits than the encoding of a motion vector of a tracked object.

It is realised that the proposed tracking algorithm has limitations. The assumption that two independently segmented objects within a defined motion tracking area of each other correspond to the same moving object, has possible drawbacks. If an object moves very quickly within the video scene, it is possible that the segmented object in a future video frame may not fall within the defined motion tracking area of the currently segmented object. Clearly an obvious solution would be to ensure the defined motion tracking area is large enough so that vehicles travelling with modest speed do not travel beyond the defined motion tracking area between processing iterations. On the other hand, the size of the defined motion tracking area shouldn't be made too large otherwise the tracking algorithm may get confused between two moving vehicles that are close to each other.

If a very fast moving vehicle does not fall within its currently defined motion tracking area, during the time required by an executed search algorithm to determine that the buffered object no longer exists within it's last recorded position, the vehicle would be segmented further along it's trajectory. The result of this delayed re-segmentation of the object would result in the same object being buffered twice. As a consequence of this, up to three instances of an object in motion at different positions along its trajectory can be separately buffered at the same time. The tracking of such a vehicle would demand more data memory than the tracking of a slowly moving vehicle, the utilised data memory would eventually be released.

The ideal tracking situation would be a video scene in which all objects are moving reasonably slowly. This can be ensured by placing the video camera some distance from the videoed scene and ensuring the lens of the camera is set to a long focal length. In this case, moving objects within the scene would appear relatively small and have low velocities. This limitation reiterates the application specific nature of the thesis project.

The proposed algorithm places no user-defined limit to the number of objects that can be segmented and tracked simultaneously. The only limitation is posed by the available data memory of the processing system.

3.3.5 *Object Decoding*

After receiving an encoded data file transmitted from the encoder, the first decoding step is that of Arithmetic decoding. Following this, the Y, U and V encoded components are then extracted from the decoded output buffer and decoded individually. The decoding process is similar to that of the reference frame decoder in that the data belonging to each of the object's 8x8 macroblocks are extracted and initially run-length decoded. The coefficients of the 8x8 macroblocks are then dequantised with reference to the JPEG reference quantisation tables. The macroblock coefficients are transformed back into the spatial domain by application of a 2D IDCT. The chrominance coefficients are upsampled from 4:2:0 ratio to 4:4:4 by the interpolating the missing pixel values in a similar fashion to that utilised during the decoding of the reference frame. Once the object's chrominance components have been interpolated, they are buffered.

In terms of controlling the buffered objects, the encoder does not transmit control headers to the decoder instructing it to create new buffers for new objects and delete existing buffers for those objects no longer tracked. During subsequent processing iterations, if the object is successfully tracked, then the encoder transmits either motion vectors or updated YUV data belonging to the object to the decoder. In the event that the encoder no longer tracks the object, the encoder does not transmit any data pertaining to that object to the decoder. On detecting the absence of data belonging to that particular object, the decoder deletes the object. This proposed method of communication between the encoder and decoder is chosen as it does away with the need of transferring actual control bytes therefore decreasing the output bit-rate. It obviously relies on the assumption that the channel over which data between encoder and decoder is transferred is error-free.

3.3.6 *Video Frame Display*

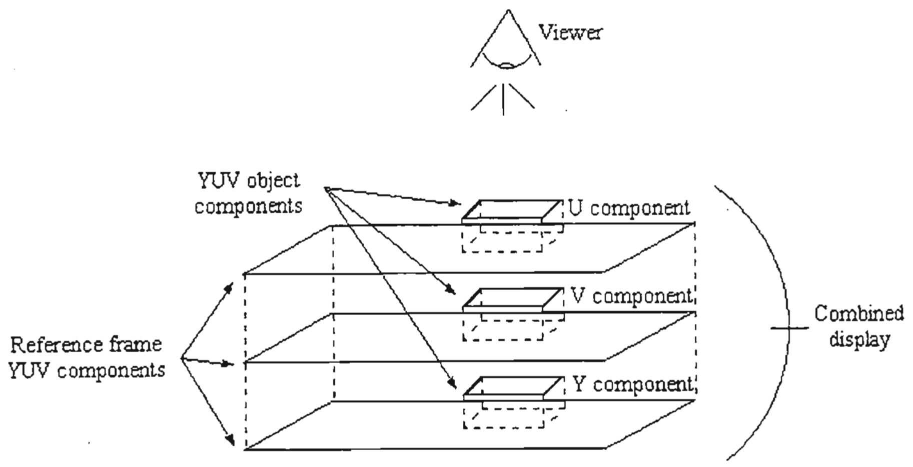


Figure 3-15: Combination of reference frame and object's YUV components.

Following each processing iteration, the decoded YUV components of the buffered reference frame are copied into three defined Y, U and V 'working' buffers. The purpose of the working buffers is to allow the combination and manipulation of both reference frame data as well as object data without destroying the original versions. The YUV components of each object are superimposed onto the reference frame by coping their respective components into the three working buffers. In effect, each superimposed component of the object overwrites the respective portion of the reference frame. An example of a synthetically created video frame is illustrated in Figure 3-15. The resulting video frame is then displayed to the system operator as part of the decoded video sequence.

3.4 Summary

This chapter describes the design of the proposed application specific, low bit-rate video compression system that is geared towards the tracking of moving objects within a video scene. The idea of the system is to superimpose segmented objects onto a static background scene and display this synthetically created video scene to the system operator. Instead of encoding input video on a frame-by-frame basis as in the case of traditional video compression standards, the proposed system encodes a selected background scene and transmits it to the decoder where it is decoded and buffered. Thereafter, input video frames are processed in order to detect objects in motion within the video scene. Detected objects are segmented, encoded and buffered by the encoder before being transmitted to the decoder. On reception, the decoder decodes the segmented object data and superimposes each object according to its defined positional offset onto the static background scene. The created video scene constitutes a single video frame within the decoded sequence that is displayed to the system operator. The segmentation process continues with the encoder tracking the motion of existing segmented objects. Low bit-rate video is achieved by calculating and transmitting motion vectors that represent the positional offset of each segmented object between a current input video frame and a previous video frame.

The background video scene, termed a reference frame, is encoded in a similar fashion to the JPEG still image compression standard. The significant difference of the proposed reference frame encoder is that it attempts to optimise encoding efficiency by sub-dividing the Y component of the YUV reference frame in macroblocks of varying size before encoding. Unlike JPEG, the proposed encoding system analyses the spatial detail of each initial 32x32 macroblock, termed a Superblock, by calculating its respective standard deviation. If the calculated standard deviation exceeds a user-defined threshold, then it is assumed that that Superblock encloses an area of the reference frame high in spatial detail. In this case, the Superblock is quad-tree partitioned into four 16x16 macroblocks in an attempt to improve the fidelity of the decoded video frame. The standard deviation of each 16x16 macroblock is also calculated resulting in each macroblock possibly quad-tree partitioned into four 8x8 macroblocks. The sub-divided Superblock is then encoded by transforming each macroblock into the frequency domain by means of a fast 2D DCT. The number of retained high frequency macroblock coefficients is limited by a user-defined setting. The purpose of this setting is to improve the efficiency of the run-length encoder that is applied after quantisation. The U and V components are then sub-divided into fixed macroblocks of size 16x16 pixels before encoding similarly to the Y component. The YUV components are then Arithmetically encoded and finally transmitted to the decoder where they are decoded and buffered.

The proposed object segmentation system is based on cluster segmentation. This process involves the construction of a difference map from the difference of consecutive YUV input video frames. The YUV difference maps are sub-divided into 8x8 macroblocks, with the mean values of each macroblock being thresholded before mapping across to a cluster map. Thereafter, the cluster map elements are grouped together to form clusters. Noise clusters are filtered from object clusters by thresholding. The co-ordinates of each cluster block are then mapped to the original input video frame. The YUV components belonging to the segmented objects are extracted from the current input video frame and are buffered. The buffered object data is then encoded and transmitted to the decoder where it is decoded and buffered similarly to the encoder.

Tracking of segmented objects is achieved by matching the position of segmented object within the current input video frame to its counterpart that was segmented in a previous video frame. A motion vector representing the positional offset of the currently segmented object with respect to its past segmented position is calculated and transmitted to the decoder. The decoder then superimposes the buffered object onto the buffered reference frame according to its motion vector. The created video frame is then displayed to the system operator as part of a decoded video sequence.

This chapter has described the theory behind the proposed low bit-rate video compression system. The following chapter describes the implementation of the proposed system and discusses the relevant design decisions that were made.

CHAPTER 4

SYSTEM IMPLEMENTATION

Chapter 3 described the theory behind the proposed low bit-rate video compression system. The proposed system performs two main tasks, namely the encoding and decoding of a reference frame, and the segmentation and tracking of objects within a video scene. Reference frame encoding and decoding is performed similarly to the JPEG still image compression standard. The significant difference between the proposed algorithm and the standard is instead of sub-dividing an image into fixed sized macroblocks comprising 8x8 pixels, the algorithm sub-divides an image (video frame) into macroblocks that vary in size depending on their spatial detail. The object segmentation and tracking algorithm is based on cluster segmentation. It is applied to the difference of two consecutive video frames. The result is that similar valued clusters are grouped into rectangular cluster blocks that effectively represent objects in motion within the video scene. The segmented object cluster blocks are then tracked offering low output bit-rates as motion vectors representing the displacement of objects between consecutive video frames are transmitted to the decoder. After superimposing the objects onto the current reference frame, the combined video scene is then displayed as a single unit to the system operator.

A significant portion of this project involves the implementation of the proposed system. The implemented system is based around the Philips TriMedia TM-1300 DSP video capture and display card. This chapter begins by describing the hardware architecture of the system and is followed by a discussion on the TriMedia DSP regarding its construction and operation. The remainder of the chapter describes the implementation of the system software. The software building blocks that constitute both the encoder and decoder are discussed in context of the complete system. In some cases, preliminary test results are presented as during the implementation stage, they helped identify potential design inadequacies of the proposed system. As a result, some of the original algorithms have been revised.

4.1 System Architecture

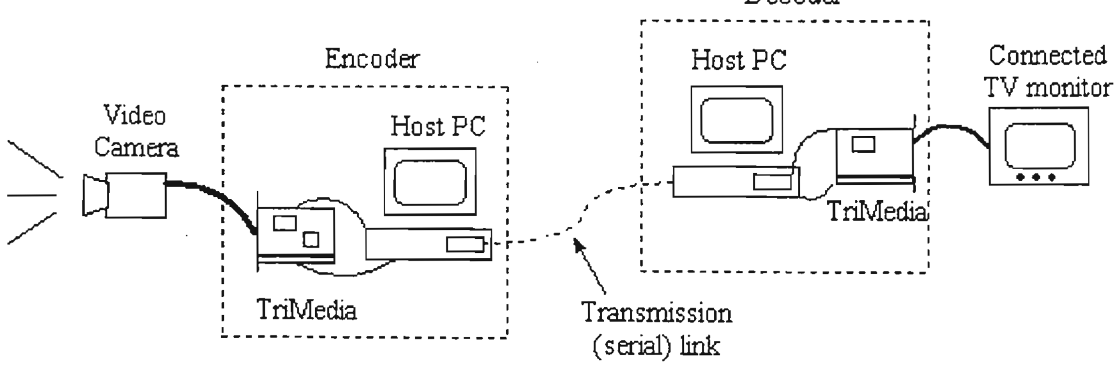


Figure 4-1: Ideal video compression system hardware architecture.

Figure 4-1 portrays the ideal hardware architecture of the video compression system to be implemented. The system comprises a separate encoder and decoder connected to each other by a serial cable as well as a source video camera and TV monitor. It was decided at the outset to utilise serial comms as the mode of communication between the encoder and decoder purely for the sake of simplicity. At the heart of both the encoder and decoder is a Philips TriMedia TM-1300 DSP video capture and display card. The TriMedia is a PCI card that comprises a standalone DSP processor with a video front and back-end. Each TriMedia card is plugged into a *host PC*. The architecture and functionality of the TriMedia card is examined in greater detail in Section 4.2.

The function of the encoding TriMedia is to capture and encode streaming input video from the video camera. The encoded video data is then serially transmitted to the decoding TriMedia where it is decoded and displayed on the connected TV monitor. The TriMedia TM-1300 has however not been designed with it's own communications port, therefore in order to transfer data between the encoder and decoder, the TriMedia has to make use of its respective host PC's communication ports.

Although ideal, the system illustrated in Figure 4-1 could not be implemented. The reason for this is that during the development stage, only one TriMedia DSP card was available. In order to overcome this limitation, it was decided to combine the encoder and decoder together. In other words, the combined system would comprise the host PC, a single TriMedia card and a video camera. The combination of the encoder and decoder is possible considering both the TriMedia and its host PC have separate processors. One of the significant advantages of the TriMedia is its ability to execute software independently of its host PC. Therefore it is possible to concurrently execute independent programs on the TriMedia and the host PC and simultaneously transfer data between the two programs in real-time. It was therefore decided to implement the TriMedia as the video encoder as its front-end is required to capture and digitise the input from the video camera. The host PC would therefore be used to decode and display the video stream. The advantage of this configuration is that an external TV monitor is no longer required. A further advantage of using the host PC as the decoder is that the displayed decoded video can be incorporated into a graphic user interface (GUI) that allows the system operator to configure the system's compression variables and view the resulting rate metrics and decoded video stream at the same time. It must however be stressed that this system configuration is suited to the development, test and demonstration environment only. If this particular system had to be commercially produced, its final configuration would resemble that illustrated in Figure 4-1 where ideally the communication link would be wireless.

The host PC utilised in this project was a desktop Dell PC comprising a 677 MHz motherboard with 128 Mbytes of RAM. The video camera was a Sony DCR-TRV11E digital Handycam.

4.2 TriMedia TM-1300 DSP Card

The Philips TriMedia TM-1300 DSP card (termed 'TriMedia' from here on) is a programmable development platform that plugs into a PCI slot of a Windows based host PC [TriMedia]. A block diagram of the TriMedia is illustrated in Figure 4-2. Although the TriMedia is able to process video, audio and telecommunication data, the proposed system is concerned with the compression of video only and thus the TriMedia's extended functionality is ignored. The front-end of the TriMedia consists of an ADC. This converter has the ability to convert either S-Video or CVBS analogue video in NTSC or PAL formats to output digital video. Input analogue video is captured and digitised on a frame-by frame basis with the digitised output video converted, by default, to YUV colour format. The U and V colour components are automatically down-sampled according to the 4:2:2 ratio. Each digitised YUV pixel comprises three 8 bit values in the range [0, 255]. The output resolution of the digitised video stream can be selected between full PAL resolution, which is default, or CIF resolution. The selection between the two output resolutions is user defined. Digitised video is then passed to the DSP for processing.

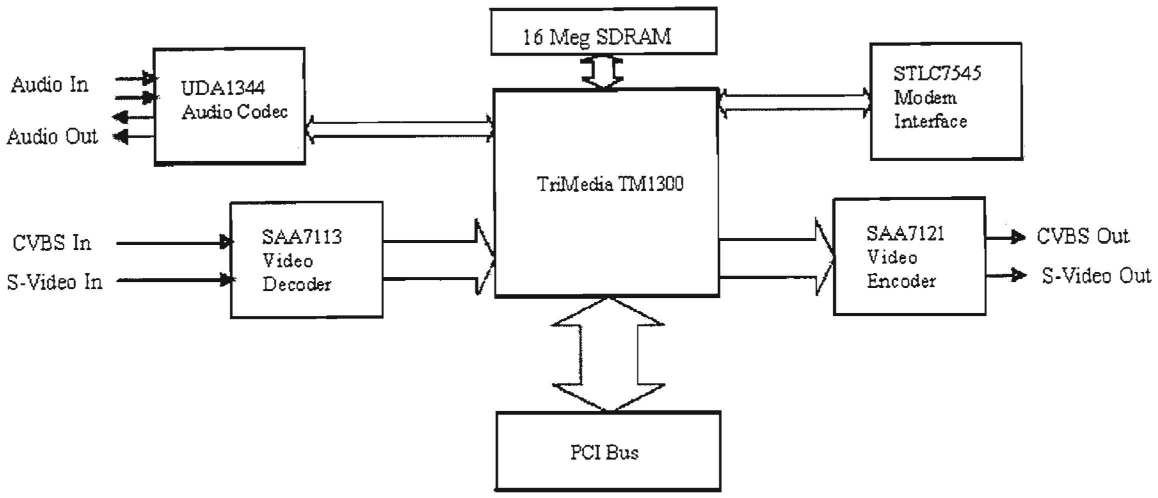


Figure 4-2: Block diagram of the Philips TriMedia TM-1300 DSP card [TriMedia].

The Philips TM-1300 DSP is optimised for multimedia type applications. It is a standalone processor that has a very long instruction word (VLIW) 32-bit floating-point architecture and is able to execute multiple operations in parallel. Peak execution rates up to 500 million operations per second (MOPS) are possible. There are 16 Mbytes of synchronous dynamic RAM (SDRAM) onboard the TriMedia for video storage and processing purposes. As an example of the more than adequate buffering ability, the onboard SDRAM can buffer up to 110 YUV video frames in CIF resolution at any one time.

The back-end of the TriMedia comprises a digital video encoder that converts digital YUV video to either S-Video or CVBS analogue video in NTSC or PAL formats. The output video can be viewed on a connected TV monitor. During conversion, the U and V colour components are automatically upsampled from 4:2:2 colour ratio to 4:4:4.

The TriMedia can be programmed in either the C or C++ languages. In this project, the software for the TriMedia was written in C. An accompanying software development environment (SDE) is used to compile and upload executable programs to the TriMedia via the PCI bus of the host PC. The SDE also provides a range of sample software programs, plus tools for debugging, simulation and performance analysis of compiled code. There is also a library of software API's.

4.2.1 Operation of the TriMedia

The section of encoder software that dealt with the capture and digitisation of streaming input video was adapted from a sample program that was provided with the SDE. The purpose of the sample program was to demonstrate the functionality of the TriMedia. On execution, the program captures streaming input PAL video from a video camera, digitises it, then reconverts it to analogue PAL video and displays it on a connected TV monitor. The output video is displayed in full PAL resolution for a few seconds and then in CIF resolution before the program terminates. The functions of the sample program pertaining to the capturing, digitisation and conversion of the output video to CIF resolution were used. The operation of the adapted software is as follows: streaming input analogue PAL video is captured and digitised at a frame rate of 25 frames/second. The digitised video is scaled to CIF resolution and converted to YUV colour space with the U and V components, by default, sub-sampled according to the 4:2:2 ratio. Once an input video frame has been digitised, an interrupt is triggered signalling the availability of the frame for processing purposes.

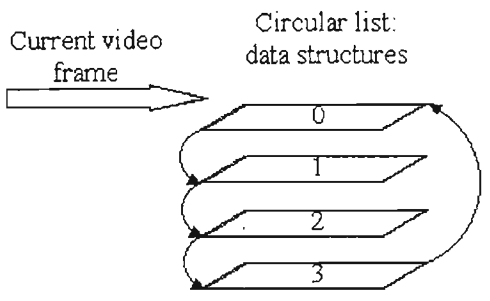


Figure 4-3: Digitised video frames buffered in a circular list.

Each newly digitised video frame is passed to the DSP in one of four defined data structures. The data structures are arranged in a circular list so that when a new video frame is digitised, the data structure at the bottom of the stack moves to the top and is used to buffer the new video frame. This process is illustrated in Figure 4-3. As future video frames are digitised, the structures within the stack are cycled in a top to bottom direction. Thus from a processing point of view, access to a newly digitised video frame as well as the past three video frames is possible.

Each data structure comprises three sections, one for each of the Y, U and V components. Each section is further subdivided into a series of line buffers, one for each row of pixels. The TriMedia system places a restriction on the line buffer size as all line buffers must start on a 64 byte boundary. Since the CIF line length of 352 pixels is not a multiple of 64, the line buffers are padded with an extra 32 bytes to give 384 byte buffers. The extra padded length appended to the end of each row of pixels within the video frame is termed *stride*. Thus the complete buffer size for the Y component of a digitised video frame is 384x288 pixels, while that of the two chrominance components is 192x288 pixels. The effect of the added stride to each digitised video frame is easily eliminated during processing.

Although the digitised video is by default sub-sampled according to the ratio 4:2:2, it was however proposed in Chapter 3 that the encoder would operate on digital video with the chrominance components sub-sampled according to the 4:2:0 ratio. Therefore each video frame that is used for encoding purposes is further sub-sampled by omitting every odd row of U and V pixels. The original 4:2:2 video frame is not discarded however as it is used in the latter decoding stages for PSNR evaluation purposes.

An important function of the TriMedia is its ability to share data with its host PC. The term data sharing translates to a shared memory block that both the TriMedia and host PC have read/write access to. This functionality is provided by means of a software API within the SDE. The initialisation process of the shared memory block is as follows. The host PC wakes the TriMedia. Once in execution, the TriMedia creates a semaphore that is to be passed between itself and the PC. Thereafter both the TriMedia and the host PC declare a common memory space of equal size within their respective data memory areas. The TriMedia API then links up the declared memory spaces so that they are visible by both independent processors. The reading and writing of data to the shared memory space is a synchronous operation that is controlled by the semaphore. The passing of the semaphore ensures that only one processor can read and write to the shared memory space at any one time. The TriMedia API limits the shared memory block to 64 Kbytes in size.

The function of the shared memory space was to provide a means of transferring encoded video data to the decoder. It is realised that the transferral of data across the PCI bus of a PC is extremely quick in comparison to serial comms or a bandwidth limited wireless link. Therefore, in order to simulate the effect of a low bit-rate channel, data was divided into 200 byte packets and a delay was executed between the transfer of each packet from the encoder to the decoder. For example, if we were simulating a 16 kbit/second bandwidth limited channel, data packets would be transferred every 100 milliseconds. Other factors such as the elapsed encoding and decoding times were also considered when calculating rate metrics such as transferred bits/second.

4.3 Reference Frame Encoder

The encoder was written in the C programming language and in modular form. An important consideration during the development of the software functions was their versatility in the sense that certain key functions would be used to encode both reference frames as well as segmented objects. During the development stage, the encoder software was compiled and tested using the BORLAND C++ Builder version 3.0 software development environment. Once the encoding software was debugged and proved functional, it was then compiled by the TriMedia SDE and uploaded to the TriMedia for execution.

The frame encoder comprises six main software functions as illustrated in Figure 4-4. The controlling function controls the implementation of the five remaining 'working' functions. Four variables are passed to the controlling function each time the frame encoder algorithm is executed. The names, types and a description of the passed variables are presented in Table 4-1. Each function was coded as a separate C file and tested individually. On satisfactory operation, the compiled C files were linked together and the encoder was tested as a complete unit.

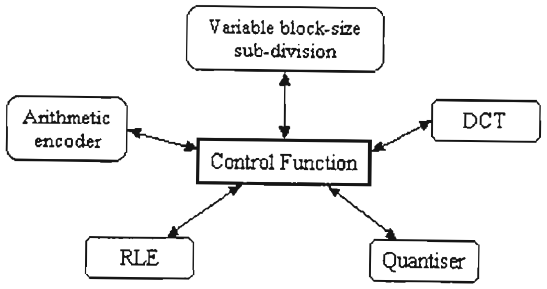


Figure 4-4: Software function structure of the implemented reference frame encoder.

Table 4-1: List of variables passed to reference frame encoder.

Passed variables		
Uchar	**Qfactor**	Quantisation factor used in association with quantisation tables
Uchar	**ref_matrix_size**	Limits the number of remaining frequency coefficients after applying the DCT
Uchar	**Thres_32**	32x32 Superblock standard deviation threshold
Uchar	**Thres_16**	Sub-divided 16x16 macroblock standard deviation threshold

Table 4-2: Reference frame encoder global variables.

	General global variables	
Uchar	**header**	Each Superblock has a header byte, used to indicate it's sub-divided structure
Uchar	**depth**	Variable used in conjunction with **ref_matrix_size** during the application of the DCT
Uchar	**Block_division[4]**	Used in conjunction with **header** in order to keep track of each sub-divided Superblock
	DCT processing matrices	
Uchar	**YSuperblock[32][32]**	Y component copied from current input video frame and placed here before encoding
Uchar	**USuperblock[16][16]**	U component copied from current input video frame and placed here before encoding
Uchar	**VSuperblock[16][16]**	V component copied from current input video frame and placed here before encoding
Float	**YOutput_matrix[32][32]**	2D Matrix contains DCT transformed Y component Superblock coefficients
Float	**UOutput_matrix[16][16]**	2D Matrix contains DCT transformed U component 16x16 macroblock coefficients
Float	**VOutput_matrix[16][16]**	2D Matrix contains DCT transformed V component 16x16 macroblock coefficients
	Input/Output buffers	
Uchar	**Ycurr_frame[110600]**	Buffered Y component of captured input video frame to be encoded (384x288)
Uchar	**Ucurr_frame[27700]**	Buffered 4:2:0 U component of captured input video frame to be encoded (192x144)
Uchar	**Vcurr_frame[27700]**	Buffered 4:2:0 V component of captured input video frame to be encoded (192x144)
Int	**YOutput_Array[101400]**	Buffer contains Y component of encoded reference frame before Arithmetic encoding
Int	**UOutput_Array[25400]**	Buffer contains U component of encoded reference frame before Arithmetic encoding
Int	**VOutput_Array[25400]**	Buffer contains V component of encoded reference frame before Arithmetic encoding
Uchar	**Encoded_Output[50000]**	Buffers contains encoded reference frame, contents to be transmitted to decoder
	Indication variables	
Int	**Encode_Out_Size**	Indicates the size of the complete encoded reference frame
Int	**Process_Time**	Keeps track of total time taken to encode reference frame (in milliseconds)

The original frame encoder passed variables between the called software functions. However on revising the design, it was decided that as the majority of the variables were in constant use by the different functions, it would make more sense from an efficiency point of view to declare the commonly utilised variables globally. The improved performance as far as the revised implementation was concerned was ratified by means of the profiler analysis program that was provided in the TriMedia SDE. The declared global variables utilised by the reference frame encoder are listed in Table 4-2.

The sub-sampled 4:2:0 YUV components of the current video frame are made available to the encoder controlling function in the **Ycurr_frame, Ucurr_frame** and **Vcurr_frame** buffers respectively. The controlling function begins by initialising all global variables to zero as well as starting the encoder processor timer. Each of the following sections describes the implementation of the respective functions, as depicted in Figure 4-4, that are called in turn by the encoder controlling function.

4.3.1 Variable Block-Size Sub-Division

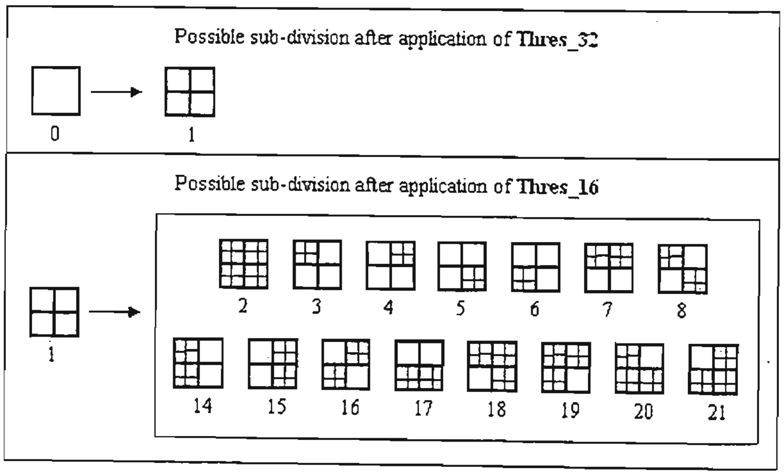
The first encoding step is the sub-division of the YUV components of the reference frame into macroblocks. Ignoring stride, the Y component of the current digital CIF video frame is sub-divided into 99 32x32 Superblocks in total while the U and V components are also sub-divided into 99 macroblocks but in this case the macroblocks are of size 16x16 pixels. This sub-division corresponds to all three YUV components being 11 blocks wide and 9 blocks high. Corresponding sets of YUV blocks are encoded separately before moving onto the adjacent set of YUV blocks. For example, the top left Y component Superblock and the top left U and V component 16x16 macroblocks comprise a set, and this set of corresponding blocks is encoded before encoding the next set of corresponding blocks to the right. The sets of blocks within the reference frame are encoded in a left to right order from the top of the reference frame towards the bottom. Locally declared counters within the encoder controlling function keep track of the position of the current set of YUV blocks being operated upon.

The 2D DCT was the first function that was coded at the outset of this project. The function required a 2D matrix of input unsigned characters while the transformed resulted was a 2D matrix of floating point coefficients. Not wanting to change the fully operational function when it was combined with the remaining encoder functions, it was decided to make use of a further set of buffers that would be used as an input for the DCT function. Thus the YUV pixels of the current frame are mapped from their 1D arrays to the respective 2D **YSuperblock**, **USuperblock** and **VSuperblock** buffers ready for cosine transformation.

Recall the discussion from Chapter 3 that the larger the size of the macroblock to be encoded, the higher its energy compaction generally at the expense of decoded fidelity. On the other hand, the smaller the size of the macroblock, the greater its retained decoded fidelity but at a decrease in energy compaction. In an attempt to retain reasonable levels of decoded reference frame fidelity, Y Superblocks may be sub-divided into smaller macroblocks during encoding. The sub-division decision is based on the measured spatial detail. Chrominance macroblocks are never sub-divided and are thus encoded at their default size of 16x16 pixels.

The spatial detail of a Y Superblock is measured by calculating its standard deviation according to (4-1) where n is the 1D size of the macroblock (in the case of a Superblock, n = 32) and $\bar{x}$ is it's mean. The calculated standard deviation is then thresholded against the variable **Thres_32**. If the standard deviation exceeds **Thres_32**, then the Superblock is quad-tree sub-divided into four 16x16 macroblocks. The standard deviation of each of the four 16x16 macroblocks is then evaluated but in this case, n is set to 16. The resulting standard deviations of the 16x16 macroblocks are thresholded against variable **Thres_16**. Once again, if a macroblocks calculated standard deviation exceeds **Thres_16**, then that particular macroblock is further quad-tree sub-divided into four 8x8 macroblocks.

$$s = \sqrt{\frac{1}{n^2-1}\sum_{i,j=0}^{n-1}\left(x_{i,j}-\bar{x}\right)^2} \tag{4-1}$$



Possible sub-division after application of Thres_32
0
1
Possible sub-division after application of Thres_16
1
2
3
4
5
6
7
8
14
15
16
17
18
19
20
21

Figure 4-5: Possible sub-divided structures of a Y Superblock.

The sub-divided structure of each Superblock is recorded for decoding purposes by means of an accompanying **header** byte. Each Superblock's **header** byte is configured according to the structure of the resulting 16x16 and 8x8 macroblocks after application of the variable block-size sub-division algorithm. Figure 4-5 portrays the structures of a sub-divided Superblock and their respective **header** values. From Figure 4-5, it is noted that the header values from 9 to 13 have been omitted. The reason for this is that during the preliminary testing stage, encoded frames were written to HDD. The problem arose that the values 9 to 13 correspond to ASCII control bytes, and thus when attempting to read the encoded frame from the HDD, several mismatch problems were experienced hence it was decided to skip these values. Note that this was purely a design choice and has no influence on the performance of the encoder at all.

4.3.2 Discrete Cosine Transform

The Y Superblock and the two chrominance 16x16 macroblocks are transformed into the frequency domain by the application of a 2D DCT. The implemented DCT equation takes advantage of the separability property of the 2D DCT by applying a fast 1D DCT to each row and column of a 2D macroblock. The implemented 1D DCT equation is based on that proposed by [Chen77]. The original 1D DCT equations proposed by [Chen77] are applicable to input arrays that have 4, 8, 16 or 32 coefficients. The 1D DCT equations are bi-directional in the sense that application in the one direction results in the forward DCT of an input array while application in the opposite direction results in the IDCT. The 1D DCT equations are presented in the form of a signal-flow chart. The signal flow chart is presented in Appendix B. As an example of the implemented 2D DCT, consider the transformation of a 16x16 U macroblock. A fast 1D DCT is applied to each row of the input 2D **USuperblock** in a top to bottom direction. Thereafter, a 1D DCT is applied to the columns of the **UOutput_matrix** buffer in a left to right direction. The resultant transformed floating point coefficients are buffered in **UOutput_matrix**.

Recall the coefficient limiting algorithm proposed in Section 3.3.1. The aim of the algorithm is to limit the number of retained low frequency coefficients after application of the 2D DCT algorithm in order to optimise output bit-rate. This can be achieved, as subsequent to transformation, there are a limited number of coefficients per macroblock to entropy encode. Considering the JPEG standard on the other hand, the number of coefficients to be entropy encoded per macroblock varies depending on the number of non-zero valued high frequency coefficients. Therefore it would be expected that encoding by means of the coefficient limiting algorithm is more efficient as a maximum number of coefficients per macroblock would be anticipated in all cases.

Considering a transformed 2D matrix, the coefficients that are retained are concentrated in the top left corner, i.e. the low frequency coefficients. The specified number of coefficients to be retained after transformation is determined by the user-defined variable **ref_matrix_size**. Recall the possible values of the **ref_matrix_size** user-defined variable include {1, 3, 6, 10, 15, 21, 28 and 36}. It would prove computationally wasteful to apply a 1D DCT to every row and column of an input macroblock considering that only a select number of coefficients are to be utilised after transformation as the unwanted DCT coefficients are simply discarded. Therefore, in order to optimise computational efficiency, the implemented algorithm applies a 1D DCT to each macroblock row as normal, however, instead of applying a 1D DCT to every column of the macroblock, a 1D DCT is only applied to those columns in which the wanted coefficients fall.

As an example, assume the user-defined variable **ref_matrix_size** has the value 21 and an 8x8 macroblock is to be transformed. According to the implemented coefficient limiting algorithm, only 21 coefficients are retained after transformation and these are concentrated around the DC (top-left) coefficient of the macroblock. The 21 retained macroblock coefficients are organised as follows: the top row of the macroblock comprises six retained coefficients, the second row comprises five coefficients, the third row comprises four coefficients and so on until the sixth row of the macroblock contains only one transformed coefficient. Figure 4-6(a) provides an example of the coefficient limiting algorithm with **ref_matrix_size** set to 21. As described previously, in order to transform the 8x8 macroblock and retain the specified number of coefficients in the most computationally efficient way possible, a 1D DCT is initially applied to each row of the 8x8 macroblock as illustrated in Figure 4-6(b). Thereafter a 1D DCT is applied only to the first six columns of the partially transformed macroblock. This is illustrated in Figure 4-6(c). All remaining unwanted coefficients are set to zero. If **ref_matrix_size** was set to 15 for example, then following the transformation of the macroblock rows, a 1D DCT would only be applied to the first five columns of the macroblock.

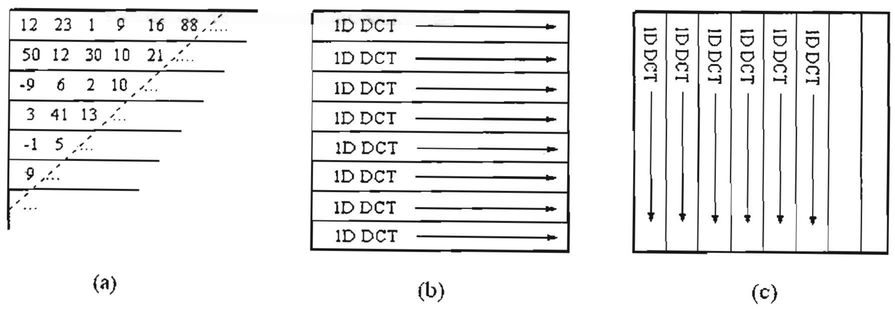


Figure 4-6: Example of the implemented optimised fast 2D DCT algorithm.

$$Additions \;=\; \left(\frac{3N}{2}\right)\left(\log_2 N - 1\right)+2 \tag{4-2}$$

$$Multiplications \;=\; N\log_2 N \;-\frac{3N}{2}\; +4 \tag{4-3}$$

Table 4-3: Comparison of the required calculations for the transformation of macroblocks with and without the coefficient limiting technique.

ref_matrix _size	Variable block size											
	8x8				16x16				32x32			
	Complete		Limited		Complete		Limited		Complete		Limited	
	Add	Mult	Add	Mult	Add	Mult	Add	Mult	Add	Mult	Add	Mult
1			234	136			1,258	748			6,402	3,828
3			260	160			1,332	792			6,596	3,944
6			286	176			1,406	836			6,790	4,060
10	416	256	312	192	2,368	1,408	1,480	880	12,416	7,424	6,984	4,176
15			338	208			1,554	924			7,178	4,292
21			364	224			1,628	968			7,372	4,408
28			390	240			1,702	1,012			7,566	4,524
36			416	256			1,776	1,056			7,760	4,640

[Chen77] provided (4-2) and (4-3) as a measure of the number of addition and multiplication calculations that would be required on applying the proposed 1D DCT to an array of dimension *N*. From (4-2) and (4-3), Table 4-3 compares the total number of additions and multiplications that are required to transform a complete 2D macroblock of varying size, to the total number of calculations that are required to transform similar sized macroblocks but with the coefficient limiting algorithm implemented. It can be seen that the required number of additions and multiplications is greatly reduced on implementing the coefficient limiting technique, especially when transforming larger sized macroblocks.

4.3.3 *Quantisation*

Following application the DCT, the controlling function calls the quantisation function. The function quantises the coefficients within the 2D **YOutput_matrix**, **UOutput_matrix** and **VOutput_matrix** buffers. The objective of quantisation is to convert each transformed 32-bit floating-point coefficient within the respective buffers to 8-bit signed characters that fall within the range [-128, +127]. Each coefficient is quantised according to:

$$g(i,j) = \left\lfloor \left(\frac{h(i,j)}{Q * Qtable(i,j)} \right) + 0.5 \right\rfloor \tag{4-4}$$

where ***h(i, j)*** are the transformed floating-point coefficients, ***Q*** is an evaluated quantisation parameter and ***Qtable(i, j)*** is quantisation coefficient obtained from a look-up table. Adhering to the JPEG standard, the quantisation parameter ***Q*** is calculated according to a user-defined **Qfactor** variable. **Qfactor** has a range [1, 100] where 100 corresponds to the least amount of quantisation and 1 the most. With regard to (4-4), the utilised quantisation parameter ***Q*** is evaluated according to the following psuedocode [Berc98]:

```
if Qfactor == 100, then Q = 1;
else if Qfactor < 50, then Q = 5000/Qfactor;
else Q = 200 – (Qfactor)*2;
```

Considering the quantisation coefficients given by ***Qtable(i, j)*** used in (4-4), the JPEG standard provides only two reference quantisation tables for macroblocks of size 8x8 pixels. Considering the proposed system utilises macroblocks of variable size, obviously these reference tables would not prove suitable for macroblocks that are larger than 8x8 pixels. Attempts were made to locate suitable quantisation tables for larger sized macroblocks, especially on the Internet, however these resulted in little success. Therefore it was decided to interpolate the coefficients of the respective quantisation tables. It is realised that the interpolated quantisation tables are not optimised with respect to the HVS, it was however felt that for the purposes of this project, they would be adequate. The interpolated quantisation tables are illustrated in Figure 4-7.

The quantisation coefficients of the interpolated 16x16 and 32x32 tables are based on the JPEG reference quantisation tables. With regard to the interpolated 16x16 quantisation table, the idea was to retain as much detail within the low frequency coefficients that remain after application of the coefficient limiting algorithm. The concept may sound illogical considering that a large percentage of the high frequency coefficients have already been discarded hence the spatial detail within the macroblock has been irreversibly lost. However a 16x16 macroblock does not mean that it contains no spatial detail, it just contains less detail on average compared to an 8x8 macroblock. This was determined by standard deviation. Therefore by lowering the AC coefficients of the 16x16 quantisation table, the output bit-rate would still be optimised and an improved amount of spatial detail would be retained compared to a table whose AC coefficients were higher.

The interpolation process that was applied was that each AC coefficient was set to approximately half of its corresponding coefficient within the 8x8 JPEG reference tables. Similarly, the 32x32 quantisation table is interpolated from the 16x16 table. The DC coefficients were calculated somewhat differently. From the original 2D DCT equation given by (2-7), the DC coefficient of a macroblock can be calculated according to:

$$DC = \frac{1}{M}\sum_{i=0}^{M}\sum_{j=0}^{N} h(i,j) \qquad (4\text{-}5)$$

where h(i, j) represents a spatial macroblock coefficient and in this case $M = N$ as the macroblock is square. In the extreme situation where all of the coefficients are equal to 255, and considering a 16x16 macroblock where M = 16, the calculated DC coefficient is equal to 4,080. Recall the purpose of quantisation is to scale each coefficient so that it falls within the range [-128, +127], the DC quantisation coefficient is calculated at 32. Similarly the 32x32 DC coefficient is calculated at 64.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	1
14	13	16	24	40	57	1	1
14	17	22	29	51	1	1	1
18	22	37	56	1	1	1	1
24	35	55	1	1	1	1	1
49	64	1	1	1	1	1	1
72	1	1	1	1	1	1	1

(a)

32	5	5	8	12	20	25	30
6	6	7	9	13	29	30	1
7	6	8	12	20	28	1	1
7	8	11	15	25	1	1	1
9	11	18	28	1	1	1	1
12	17	27	1	1	1	1	1
24	32	1	1	1	1	1	1
36	1	1	1	1	1	1	1

(b)

64	2	2	4	6	10	12	15
3	3	3	4	6	14	15	1
3	3	4	6	10	14	1	1
3	4	5	7	12	1	1	1
4	5	9	14	1	1	1	1
6	8	13	1	1	1	1	1
12	16	1	1	1	1	1	1
18	1	1	1	1	1	1	1

(c)

32	9	12	23	48	48	48	48
9	10	13	33	48	48	48	1
12	13	28	48	48	48	1	1
23	33	48	48	48	1	1	1
48	48	48	48	1	1	1	1
48	48	48	1	1	1	1	1
48	48	1	1	1	1	1	1
48	1	1	1	1	1	1	1

(d)

Figure 4-7: Utilised quantisation tables: (a) 8x8 luminance, (b) 16x16 luminance, (c) 32x32 luminance, (d) 16x16 chrominance.

As a further safety check, as each coefficient is quantised it is evaluated and if it exceeds the range, it is then set to the respective limit, i.e. –128 or +127.

4.3.4 *Run-Length Encoding*

The run-length encoding (RLE) function losslessly encodes the quantised DCT coefficients within the respective **YOutput_matrix**, **UOutput_matrix** and **VOutput_matrix** buffers. The algorithm starts by converting an input 2D matrix of coefficients to a 1D array by scanning the coefficients in a zig-zag pattern in order of increasing frequency, i.e. starting with the coefficient in the top left corner and working towards the bottom right. The scanning order is indicated by the sequence of numbers in Table 4-4. Note, due to the limited number of coefficients as determined by **ref_matrix_size**, a maximum of 36 coefficients are scanned per **Output_matrix**. With regard to **YOutput_matrix**, a maximum of 36 coefficients per macroblock can be scanned.

Having converted the 2D matrix to a 1D array, the actual RLE process involves scanning and counting the number of consecutive coefficients that have a similar value. The encoder is designed to substitute five or more consecutive coefficients that have a similar value with a four byte RLE sequence. The first two bytes of the RLE sequence are used to provide a unique RLE identification header while the last two bytes indicate the value of the common coefficient and the number of times that it occurs consecutively. The two-byte identification header comprises the values '1' and '127'. The encoder was designed to ensure that these two consecutive values could never occur together naturally under normal conditions thus preventing possible confusion when decoding a RLE sequence. Furthermore, it was also designed to encode only five or more consecutive similar valued coefficients as considering a RLE sequence requires four bytes, the encoding of less than five coefficients would not be worthwhile.

Table 4-4: Zig-zag scanning pattern.

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	
3	8	12	17	25	30		
9	11	18	24	31			
10	19	23	32				
20	22	33					
21	34						
35							

It is realised that the implemented RLE can never be as efficient as that of a JPEG RLE as in the case of a JPEG quantised 8x8 macroblock, it is highly likely that many high frequency coefficients will be zero-valued therefore resulting in a high efficiency of the applied RLE. In the case of the implemented system, most of the high frequency coefficients are discarded before application of the RLE, therefore it would be expected that the compression efficiency of the implemented RLE decreases as the discarded high frequency coefficients are not encoded. It is also possible that the implemented RLE may not result in any compression when applied to certain macroblocks. However considering the implemented RLE is designed in such a manner that the number of output encoded symbols can never exceed the number of input coefficients, the RLE step is incorporated into the compression system nonetheless.

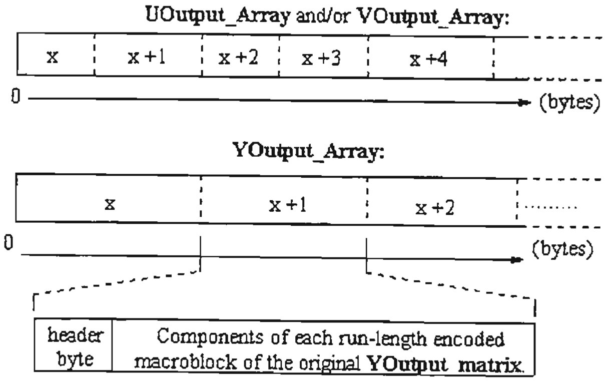

UOutput_Array and/or VOutput_Array:
x
x+1
x+2
x+3
x+4
0
(bytes)
YOutput_Array:
x
x+1
x+2
0
(bytes)
header byte
Components of each run-length encoded macroblock of the original YOutput matrix

Figure 4-8: Typical composition of the respective **YOutput_Array, UOutput_Array** and **VOutput_Array**'s.

The outputs of the RLE encoded chrominance blocks are stored as arrays. As the chrominance blocks have a fixed number of coefficients for any user defined compression setting, this value must be transmitted with the coefficients and thus there is no need to send the number of encoded values to the decoder as well. As far as the Y Superblock is concerned, each separate macroblock is treated independently in exactly the same manner. In order to inform the decoder of the format of the stream of values, each Y Superblock is preceded by a **header** byte that uniquely defines its sub-divided structure. Figure 4-8 illustrates the packing of the encoded components into arrays where *x* indicates the encoded data that belongs to each run-length encoded macroblock.

It was decided further that in order to reduce the number of overhead bytes that would be transmitted to the decoder, the **header** byte would also be used to indicate the number of coefficients that are retained per macroblock following application of the coefficient limiting algorithm. The combined **header** byte eliminates the need of transmitting a separate **ref_matrix_size** byte to the decoder. Table 4-5 illustrates the construction of a **header** byte from the combination of the **ref_matrix_size** variable (most significant three bits) and the five least significant bits used to define the structure of the sub-divided Superblock with the values given in Figure 4-5.

Table 4-5: Header byte construction.

ref_matrix_size value:	Resulting **header** byte:
1	000xxxxx
3	001xxxxx
6	010xxxxx
10	011xxxxx
15	100xxxxx
21	101xxxxx
28	110xxxxx
36	111xxxxx

Each of the 99 Y, U and V macroblocks of a reference frame are encoded by applying the above described encoding functions.

4.3.5 Arithmetic Encoding

Arithmetic encoding is the last encoding stage of the implemented reference frame encoder. The implemented Arithmetic encoder is based on the lossless encoder proposed by [Witt87]. The Arithmetic encoder and decoder functions were the only functions incorporated into the compression system that were not written from scratch, i.e. the source code was written by [Witt87] and merely adapted. Once each macroblock has been run-length encoded and temporarily buffered, the Arithmetic encoding function is called. The function encodes the Y, U and then V components with the result being buffered in the **Encoded_Output** 1D array before being transmitted to the decoder.

The first seven bytes of the **Encoded_Output** buffer are utilised as configuration bytes for decoding purposes. The first byte holds the quantisation parameter and the next six bytes hold the lengths of the Y, U and V encoded output arrays. These three 16-bit values are used as counters during decoding of the reference frame's YUV components. Once the reference frame has been completely encoded, the controlling function stops the process timer and converts the elapsed time to milliseconds. The total amount of encoded data in the **Encoded_Output** buffer is recorded by the **Encode_Out_Size** variable. This variable along with the elapsed processing time in milliseconds and encoded frame is then transferred to the decoder.

4.3.6 Preliminary Testing

The correct functionality of the individual encoding functions was tested before undertaking the development of the decoder. At this stage, the encoder could only be tested on a low level basis, i.e. the execution of each software function was stepped through in the BORLAND C++ Builder ver. 3.0 debugger while a variable watch window was used to monitor the respective variables.

With regard to the testing of the Variable Block-Size Sub-Division software function, the raw YUV components of a video frame captured by means of the TriMedia were saved to the host PC's hard disk drive (HDD). A test application was written to read in and buffer the YUV components of the video frame from the HDD. The sub-division of the Y Superblocks into smaller macroblocks was tested by means of a further software test function. The test function physically drew the sub-divided block structure of each Superblock onto a copy of the Y component of the reference frame and then saved it to the HDD. Then by using an image viewer/editor such as PaintShop Pro version 4.12 for example, the saved image was visually scrutinised. Figure 4-9 illustrates a typical result that was obtained during the testing stage.

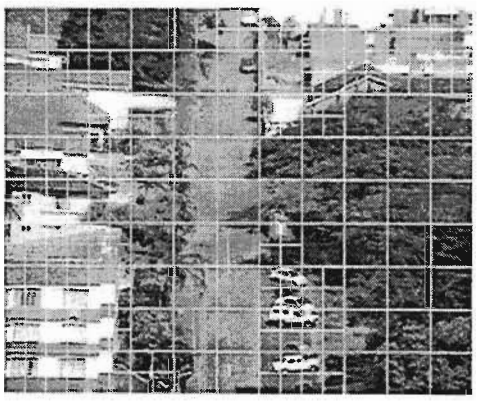


Figure 4-9: Example of a sub-divided frame.

The DCT and quantisation functions were tested by stepping through and transforming a set of input 2D matrices and then comparing the results with those obtained from similar input matrices using MATLAB ver 5.0. The remaining encoder functions were tested by individually stepping through them and monitoring the results.

4.4 Reference Frame Decoder

The reference frame decoder comprises six main software functions as illustrated in Figure 4-10, these being the inverse functions of the encoding process. All of the decoder functions were written in C programming language. The variables and arrays required by the decoder are similar to those of the encoder and as in the case of the encoder, the most commonly utilised variables were declared globally.

Once the encoded frame has been received by the decoder, the controlling function extracts the first seven bytes from the buffer and configures the quantisation parameter as well as the three 16-bit Y, U and V counters. Thereafter, the following decoding functions are called in turn.

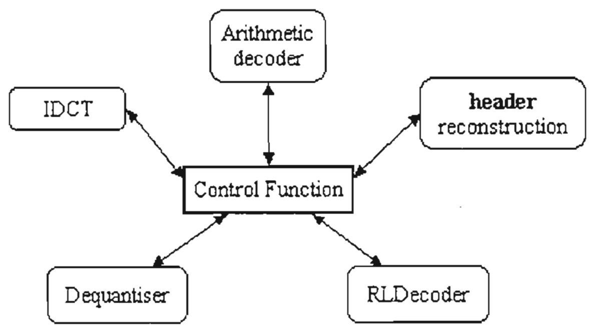


Figure 4-10: Structure of the implemented reference frame decoder software functions.

4.4.1 Arithmetic Decoder

The first decoding step is that of Arithmetic decoding. As in the case of the implemented encoder, the decoding software function is an adapted version of that authored by [Witt87]. Taking the seven-byte offset into account, the data is then Arithmetically decoded and placed in the globally declared **Decoded_Output** buffer. The result is that the decoded data is in the form of a single 1D array comprising the run-length encoded Y, U and V components of the reference frame. The three configured 16-bit counters are used to tally the number of bytes that belong to each encoded component as well as provide offset pointers to the beginning of the encoded U and V components. Following Arithmetic decoding, the following functions are called in the presented sequence a total of 99 times.

4.4.2 Header Reconstruction

Considering a single set of encoded YUV blocks that exist in the **Decoded_Output** buffer, the **header** byte for each super block is extracted and used to configure the respective global variables that define the sub-divided structure of the Y Superblock and **ref_matrix_size**. The function dissects the extracted **header** byte and uses the three most significant bits with an associated look-up table to reconstruct the original **ref_matrix_size** variable. The least significant five bits of the extracted **header** byte are used in conjunction with a further look-up table to define the sub-divided structure of the current Y Superblock.

4.4.3 Run-Length Decoding

The function performed by the run-length decoder is twofold. Firstly, it run-length decodes (RLD) the data belonging to the individual macroblocks of the Y Superblock as well as the U and V 16x16 macroblocks. At the same time, the decoded data is placed in globally declared 2D matrices ready for dequantisation and application of the inverse discrete cosine transform.

The operation of the RLD function is as follows. Consider the decoding of a U 16x16 macroblock, the RLD function executes a loop that analyses the value of each encoded byte belonging to the macroblock being decoded. If two adjacent bytes do not constitute the unique two-byte RLE identification header, i.e. '1' and '127', then the current byte being analysed is copied to a locally declared 1D array. A loop counter is incremented and the next byte is analysed. In the case of the two-byte RLE identification header being detected, the following two bytes indicating the value of the common byte and the number of times that it occurred consecutively are extracted. A further loop is then executed that writes the common value for the specified number of times to the locally declared 1D array. This process continues until the number of decoded coefficients within the locally declared 1D array equals the number of original coefficients within each macroblock as given by **ref_matrix_size**.

The run-length decoded coefficients are then sorted into the declared 2D matrix in a zig-zag pattern given by Table 4-4. This decoding process is similarly applied to the Y and V blocks. The decoded coefficients within the 2D matrices are then ready for dequantisation.

4.4.4 Dequantisation

The objective of this function is to convert the coefficients, of type signed characters, to signed 32-bit integers ready for transformation back into the spatial domain by means of the IDCT. The dequantiser makes use of the same quantisation reference tables that were used during encoding. However in this case, each macroblock coefficient is multiplied to the product of its corresponding coefficient within the quantisation reference table and the extracted quantisation parameter. The structures of the Y, U and V blocks remain unchanged after dequantisation.

4.4.5 Inverse Discrete Cosine Transform

The implemented inverse discrete cosine transform is achieved by applying the fast 1D DCT proposed by [Chen77] to the columns and then rows of a 2D macroblock in reverse direction. This method of inverse transformation is possible due to the bi-directional property of the proposed algorithm. The objective of the IDCT is to transform the respective YUV coefficients to 8-bit unsigned characters with a range of [0, 255]. Considering the IDCT of the U and V 16x16 macroblocks, the 1D IDCT is applied to each column within the respective macroblocks followed by each row. Unlike the forward DCT, processing effort cannot be optimised in terms of only applying the 1D IDCT to a select number of columns as in this case, each output coefficient is a calculated value.

The values of the resulting spatial domain coefficients are tested and corrected in order to ensure that they fall within the specified range.

4.4.6 Reference Frame Reconstruction

The final decoding step is the reconstruction of the decoded reference frame. This step involves the mapping of the contents of the decoded 2D matrices to declared 1D YUV buffers. For display purposes, the GUI requires a decoded frame to be presented in three separate unsigned character buffers of size 101,376 bytes (352x288) containing the individual YUV components sampled according to the 4:4:4 ratio.

Regarding the reconstruction of the Y component, the coefficients of each row within the Y 2D matrix are mapped to a calculated offset position within the declared 1D buffer. The frame rows are buffered one after the other. The reconstruction of the U and V reference frame components is more complicated as during the mapping process, the components are upsampled from a ratio of 4:2:0 to 4:4:4. In Section 3.3.2, it was proposed that the U and V components would be interpolated from the means of themselves and their neighbours. Considering the sample of the reference frame illustrated in Figure 4-11, the U(0, 1) and V(0, 1) pixels to be interpolated take on the calculated mean values of the

U(0, 0) and U(0, 2), and the V(0, 0) and V(0, 2) pixels respectively. The U(1, 0) and V(1, 0) pixels are calculated from the mean values of the U(0, 0) and U(2, 0), and V(0, 0) and V(2, 0) pixels respectively. Finally the pixels U(1, 1) and V(1, 1) are interpolated from the mean values of the U(0, 0) and U(2, 2), V(0, 0) and V(2, 2) pixels respectively. This interpolation process is applied to all the missing U and V pixels within the frame. As the last row and column of both the U and V components cannot be interpolated by this method, the pixel values of the adjacent row and column are copied. The rows of the upsampled reference frame U and V components are buffered one after the other similarly to the Y component.

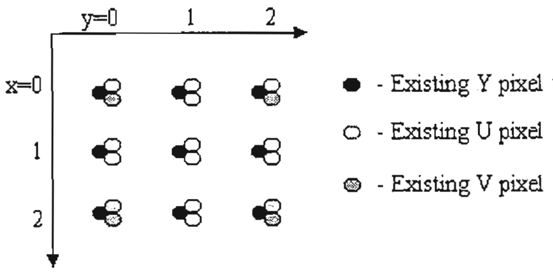


Figure 4-11: Sample of U and V pixels to be interpolated.

The described functions are executed within a loop such that each of the 99 sets of encoded Y Superblocks and U and V macroblocks are decoded. On conclusion, the individually buffered YUV components are presented to the GUI for display along with calculated rate metrics.

4.4.7 Preliminary Testing

The functionality of the decoder software was once again tested in conjunction with an encoded frame by stepping through the various functions using the BORLAND C++ Builder ver 3.0 debugger. A variable watch window was utilised in order to monitor the progress of the respective global and locally declared variables within the different functions. Regarding the functionality of the dequantiser and IDCT, the results were compared to those obtained from MATLAB ver 5.0.

As the decoder was ultimately to be executed on the host PC and not the TriMedia, the individually tested functions were compiled and linked together in the BORLAND C++ Builder SDE. The functionality of the complete decoder was tested by reading in a saved version of an encoded reference frame from the PC's HDD, decoding it and then saving the individual YUV components in three separate files to the HDD. The individual YUV components were then visually scrutinised by means of the image viewer Paint Shop Pro for unexpected artefacts.

The TriMedia SDE supplied an executable program that reads in the separate YUV components of an image that are sub-sampled according the ratio 4:2:2 from the PC's HDD, and converts the YUV image into a 24-bit colour BITMAP image. Thus a further test function was written that sub-sampled the decoded YUV reference frame from 4:4:4 to 4:2:2 and saved the individual components to the PC's HDD. The BITMAP conversion application was executed and the resulting colour BITMAP frame was visually tested once again using the Paint Shop Pro image viewer.

4.5 Object Segmentation

Similarly to the frame encoder and decoder, the object segmentation software was written in C programming language and was compiled and debugged in the BORLAND C++ Builder ver 3.0 SDE. Once the software operated satisfactorily, it was then compiled by the TriMedia SDE and uploaded to the TriMedia for further testing.

The procedure followed during the development of the object segmentation and tracking algorithms was different compared to that of the reference frame encoder and decoder. Considering the implemented frame encoder and decoder are based upon adapted versions of the JPEG still image compression standard, the development of the implemented functions was reasonably straight forward as they were based upon well established and documented algorithms. On the other hand, the development of the proposed object segmentation and tracking algorithms was not as formal as there were no definite and established algorithms that could be used to provide a sound basis. Therefore the algorithms tended to follow a development cycle that included planning, writing code, testing and then adjusting. In order to aid the following descriptions of the implemented algorithms, the preliminary testing results of the respective functions are included in order to motivate the adjustments that were made to the original proposals.

The structure of the object segmentation and tracking algorithm comprises five main functions as portrayed in the flowchart in Figure 4-12. Although the object tracking algorithm is not described in this section, it is included in the flowchart in order to clarify its relationship with the object segmentation algorithm. The implementation of the object tracking algorithm is described in Section 4.6.

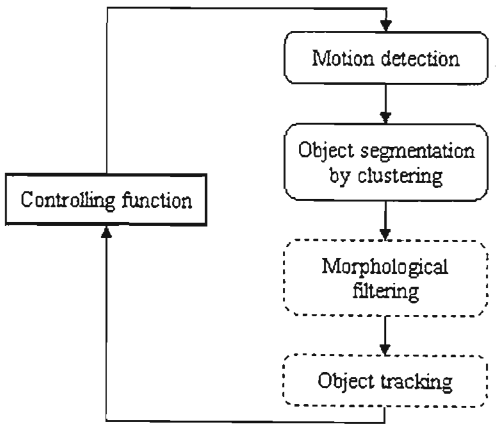


Figure 4-12: Implemented object segmentation algorithm.

The object segmentation algorithm is directly controlled by the GUI, which in turn directly controls the respective object segmentation functions. Two user-defined object segmentation variables are continually passed to the controlling function at the beginning of each processing iteration. The user-defined variables are listed in Table 4-6. A processing iteration is defined as one complete execution of the cyclical flowchart illustrated in Figure 4-12. Once an iteration has completed, the encoder transfers the compressed object data to the decoder where it is decoded and buffered. The elapsed encoding time is also transferred to the decoder which is ultimately displayed on the GUI. The transferral of the segmentation variables to the controlling function at the beginning of each processing iteration enables the system operator to adjust the algorithm on the fly.

Table 4-6: Passed object segmentation variables.

Passed variables		
int	**ClusMap_Thres**	Cluster map threshold used to filter noise in preliminary clustering step
int	**ClusBlock_Thres**	Cluster block threshold filters noise cluster blocks from object cluster blocks

Table 4-7 lists the globally declared variables that are utilised by the object segmentation software functions.

Table 4-7: Object segmentation global variables.

Cluster segmentation buffers		
uchar	**Ydiffmap[101400]**	Buffers the Y component difference map
uchar	**Yclustermap[1584]**	Buffers the Y component of the cluster map constructed from the difference map
Object co-ordinate buffers		
int	**clustercoords[801]**	Buffers the co-ordinates of segmented cluster block that is in motion
int	**Object_Coords[801]**	Buffers the co-ordinates of the filtered segmented object cluster block

The following sections describe the implementation of the individual object segmentation functions that are called in turn by the controlling function.

4.5.1 Motion Detection

The first step involved in the segmentation of an object in motion within a video scene is to detect its motion. In Section 3.3.3 it was proposed that objects in motion within a video scene would be detected by calculating a difference map between consecutive input video frames. A difference map is constructed by calculating the absolute value of the difference between corresponding pixels within consecutive input video frames. It was also proposed that a difference map for all three YUV components would be constructed in an attempt to maximise the amount of motion information conveyed within consecutive video frames.

The proposed algorithm was implemented and tested by capturing two consecutive YUV video frames by means of the TriMedia, constructing and saving the YUV difference maps to the PC's HDD. In the early investigative phase, several different approaches were tried on several different video sequences to evaluate their effectiveness and to find a suitable candidate algorithm for further work. Initial tests on differencing adjacent frames of each of the Y, U and V components showed that in the application specific scene under consideration, the motion was inadequate to produce any significant difference within a single frame time. By extending the time delay to using alternate frames for differencing, the Y map, as originally postulated, showed some relevance with the U and V maps still showing very little movement. Figure 4-13(a) provides an example of an original test video scene that was used in conjunction with its successive video frame to construct the YUV difference maps. The majority of the scene is stationary except for the vehicle moving along the road within the centre of the video scene. The constructed Y, U and V difference maps are illustrated in Figure 4-13 (b), (c) and (d) respectively.

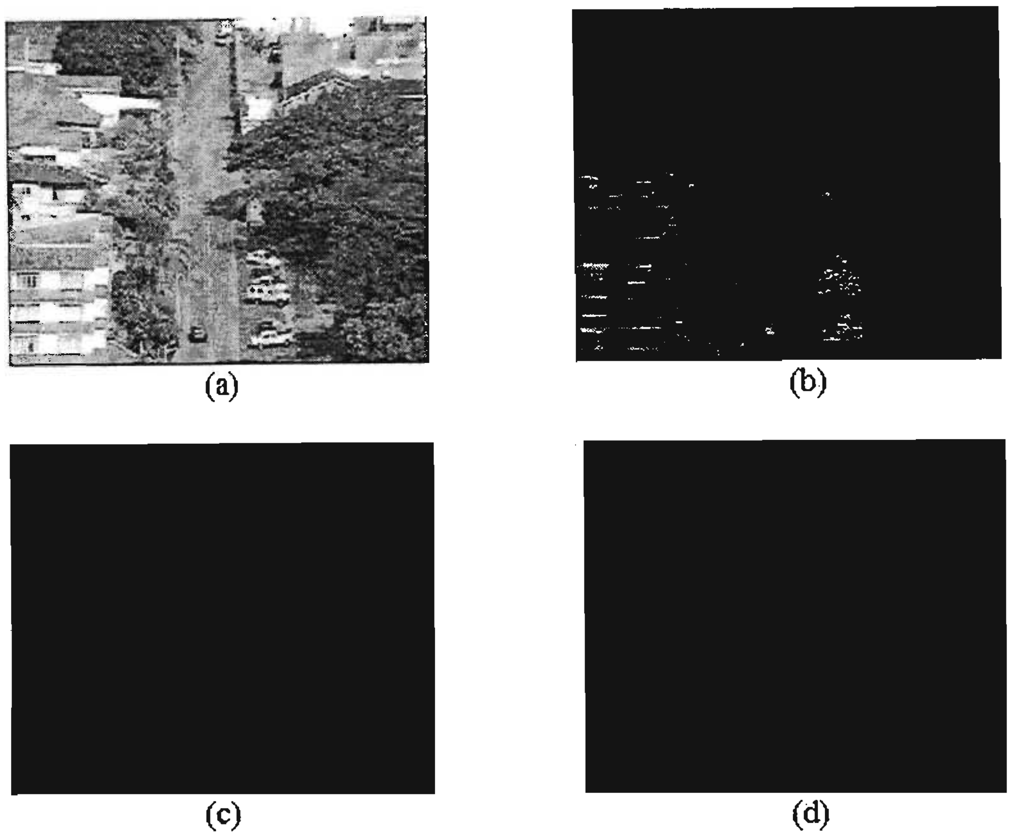


Figure 4-13: Example of test video scene and the resulting YUV difference maps.

The inability to detect slow moving vehicles by means of consecutive frame differencing is quantified by analysing a typical scenario. Assume a 200m stretch of road that is represented by 200 pixels, thus each pixel represents 1m. If a vehicle travels one pixel between consecutive frames, i.e. over a period of 40 milliseconds, the vehicle must therefore be travelling at a minimum speed of 1/40ms = 25m/s (90 km/h) for it to be detected with a difference map. This calculated speed is too high for suburban situations for which the application specific system was intended, thus the inter frame delay requires adjusting in order to detect slower moving vehicles. If the minimum speed at which vehicles are to be detected is stipulated at 20km/h, with reference to the above stretch of road, this speed corresponds to a frame delay of 180 milliseconds. The inter frame delay corresponds approximately to five frames therefore it would be expected that by differencing over five frames, vehicles travelling in excess of 20km/h would be easily detected.

The difference map in which motion is detected was therefore constructed by differencing over five frames according to

$$Ydiffmap_{i,j} \quad = \quad \left|Y_{n-2}(i,j) - Y_{n+2}(i,j)\right| \qquad (4\text{-}6)$$

Note only the Y component of the frames has been used due to the lack of motion information conveyed by the U and V components. Naturally the focal length of the video camera determines the ratio of meters represented per pixel within the video frame. Figure 4-14 illustrates this relationship by graphing the minimum number of pixels that a vehicle would have to travel over five frames in order for it to be detected as a function of the number of meters represented by each pixel within the video frame.

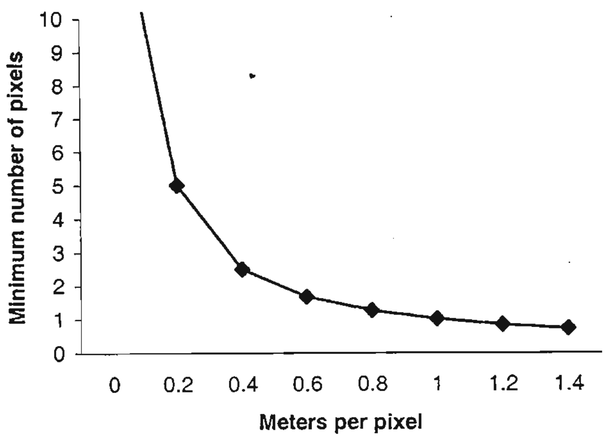


Figure 4-14: Minimum pixel relationship in order for a vehicle to be detected.

For every iteration that (4-6) is applied, every third frame denoted by Y_n is buffered. Its purpose is to provide a reference from which objects are finally extracted. The buffered video frame is here onwards termed the *target video frame*. Following completion of the implemented segmentation and tracking algorithms, the next set of five frames are received and operated upon in a similar fashion and so on.

4.5.2 Cluster Segmentation

Having constructed the Y difference map, the next step is that of segmenting the objects in motion from the video scene while at the same time filtering as much ambient noise as possible. Figure 4-15(a) provides an illustration of a target video frame of a particular test scene. The resulting difference map is illustrated in Figure 4-15(b). The motion of the vehicle in the centre of the difference map is clearly identifiable but it is however surrounded by a significant amount of ambient noise. Considering that the processed video stream originates from a digital camera, is then translated to an analogue video stream and finally digitised again at the TriMedia card, it is to be expected that there would be position shift noise where elements in the scene may appear in different pixel positions in different frames, even if the apparent movement were only a single pixel. This is going to have the most significant effect at positions of sharp intensity changes as illustrated by the building in the lower left quadrant. Digitisation jitter of a single pixel at a contrast boundary would result in a white pixel appearing in a neighbouring dark region or vice versa. To reduce the likelihood of this noise being misinterpreted as part of some moving object, some form of filtering was obviously required. The implemented filtering techniques are explained in context with the segmentation algorithm.

The implemented segmentation algorithm was based upon an adapted version of the clustering technique proposed by [Krav99]. The first step of cluster segmentation is the construction of a *cluster map*. This involves sub-dividing the difference map into 8x8 macroblocks and mapping the mean value of each macroblock to its corresponding position within the declared **Yclustermap** buffer. Considering the resolution of the difference map is 352x288 pixels, the sub-division and the mapping of the mean of each 8x8 macroblock results in the cluster map having a resolution of 44x36 pixels. The effect of constructing a cluster map is twofold. Firstly, areas of motion within a difference map generally consist of concentrated high-valued pixels compared to the low-valued pixels of non-motion areas. Thus it is expected that the calculated mean values of motion areas would greatly exceed that of non-motion areas. In such a case they could be separated by means of thresholding. Secondly, by calculating the mean of a macroblock, it has the affect of averaging out random noise.

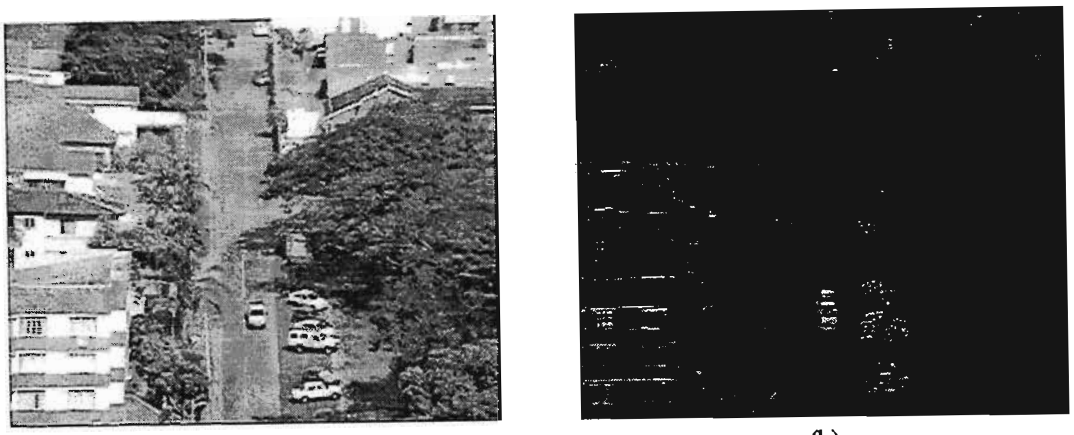


Figure 4-15: Example of target video frame and constructed difference map.

In order to take advantage of the expected difference in mean values between motion cluster blocks and noise cluster blocks, it was decided to implement a noise filter that would be effective during the construction of the cluster map. In this case, as each macroblock mean is calculated, it is thresholded against the user-defined **ClusMap_Thres** variable. Any mean value that falls below the **ClusMap_Thres** is set to zero before being mapped to the cluster map. The effectiveness of this filtering stage is analysed in Chapter 5.

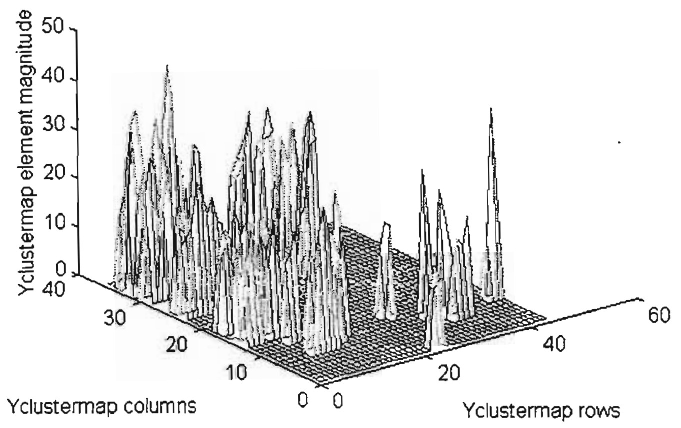


Figure 4-16: Constructed cluster map.

Figure 4-16 portrays the cluster map that was constructed from the difference map illustrated in Figure 4-15(b). In this case, **ClusMap_Thres** was randomly set to the value twelve. It is noted that the majority of the plot comprises zero-valued elements as expected however, there is still a significant amount of noise towards the top left of the map. With respect to Figure 4-15(a), this noise is concentrated in the area of the building in the lower left corner of the video frame.

In its raw form, a cluster map comprises a set of independent elements that represent the mean of a corresponding 8x8 macroblock within the difference map. In order for the independent elements to convey information, the implemented algorithm groups similar valued neighbouring cluster pixels into rectangular *object cluster blocks*. It was decided to restrict the resulting object cluster blocks to be rectangular as this is the rough shape of the vehicles being tracked. In terms of grouping similar valued neighbouring elements into cluster blocks, it was assumed that non-zero cluster elements that result due to objects in motion are concentrated within in a mostly rectangular area surrounded by zero-valued elements. This assumption is dependent on the situation in which a vehicle is moving relative to a stationary background. Therefore the goal of the cluster algorithm is to group as many non-zero, but similar valued cluster elements within a localised area together.

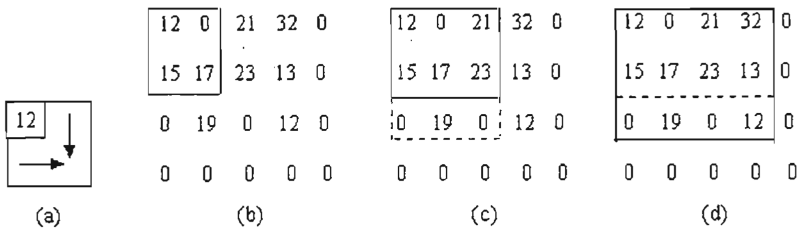


Figure 4-17: Example of cluster block growing algorithm.

The process in which cluster elements are grouped into cluster blocks from here onwards is termed cluster block growing. The operation of the implemented cluster block growing algorithm is best explained by means of an example. The algorithm starts by scanning the cluster map rows in a left to right, top to bottom order. On locating a non zero-valued element, termed the pivot element, the algorithm effectively places a square (2x2 elements) around the pivot with it located in the top left corner as illustrated in Figure 4-17(a). The objective of the first stage of the algorithm is to grow a rectangular cluster block in a downwards and rightwards direction, also illustrated in Figure 4-17(a), with respect to the pivot element. The values of the elements in the row beneath and the column to the right of the pivot element are evaluated. The criterion for the growing of the cluster block is as follows: if the number of non-zero elements within the row or column being evaluated is equal to or greater than half the total number of elements within the respective row or column, then that row or column is 'tagged' suitable for further growth in that direction during that next block growing iteration. In the situation where the current row, for example, being evaluated does not satisfy the growing criterion, then during the following growing iteration, the growing algorithm does not 'grow' the rectangular cluster block past that particular row. It still nonetheless grows past the column if it is suitably tagged. This process attempts to maximise the size of a cluster block.

Figure 4-17(b) illustrates the situation where both of the elements in the row beneath and the column to the right of the pivot element satisfy the growing criterion and are thus both tagged. Figure 4-17(c) illustrates the situation where the algorithm attempts to grow two rows beneath that of the pivot element. In this case, however, the growing criterion is not satisfied thus the cluster block is not grown downwards. The criterion is satisfied regarding the two columns to the right of the pivot element thus the cluster block is grown to the right. Figure 4-17(d) illustrates the situation where the algorithm attempts once again to grow downwards in the following block growing iteration. In this case an extra element has been added to the row by the inclusion of a further column in the previous block growing iteration. As a result, both the row and column satisfy the growing criterion and are thus tagged and ultimately included within the cluster block. During the next block growing iteration, more than half of the elements in both the row below and the column to the right would be zero-valued, thus the growing criterion is not satisfied and the growing algorithm is thus halted.

The second stage of the growing algorithm is to grow the rectangular cluster block from the same pivot element in a downward and leftwards direction in order to maximise the complete cluster block size.

This step is executed in exactly the same manner as that described above. On completion, the two cluster blocks are combined to form a single cluster block. It is however possible that the row depth (size) of the left cluster block is not the same as the right cluster block. In this case, the bottom row of the combined cluster block is subjected to the growing criterion once again. If it doesn't satisfy the growing criterion, then that row is chopped off and the row above is then evaluated. This process continues until a combined row satisfies that growing criterion. The two columns and the top row of the combined cluster block are re-evaluated according to the growing criterion in a similar fashion.

As a further noise filtering stage, the mean of the segmented object cluster block is calculated and thresholded according to the user-defined **ClusBlock_Thres** variable. The idea of this filter is due to noise being both random in value and position, hence it is expected that cluster blocks that originate from genuine vehicles in motion should generally have higher mean values than cluster blocks that result due to random noise. The effectiveness of this form of filtering is also evaluated in Chapter 5. Following the filtering stage, it is assumed that the remaining cluster blocks correspond to objects in motion and thus the top, bottom, left and right co-ordinates of the cluster blocks are recorded in the global **clustercoords** array. The algorithm was designed to prevent cluster elements being grouped into more than one object cluster block at any one time.

4.5.3 Morphological Filtering

Although this function was not expressly defined in the proposed design of the low bit-rate video compression system, it was included in order to refine the segmentation of objects. Due to the nature of the implemented algorithm, it is possible that an object as well as unwanted background scenery would be bounded within the same rectangular bounding block. The inclusion of this scenery is firstly unnecessary and secondly would have an adverse affect on the output bit-rate of the encoder as there would be more data per segmented object to encode. Therefore we need to separate the relevant object from its unwanted surroundings. The method used for this purpose was morphological opening. This process involves the erosion followed by the dilation of a pixel based on its neighbours. A more in-depth description of morphological opening was provided in Chapter 3.

Morphology is traditionally performed on binary images. When considering groups of pixels within such an image, morphological opening effectively filters isolated non-zero pixels while at the same time merges non-zero adjacent pixels into groups. The implemented filtering process involves the mapping of the co-ordinates of a rectangular cluster block to the original difference map followed by the application of morphological opening. Considering the original object within the difference map, once the filtering has completed, it would be expected that the shape of the object would be better defined as the scattered fringe pixels surrounding the object would be filtered and the pixels forming the body of the object would be grouped tightly together. The only difference between the implemented opening technique and the traditional one was that it was adapted to cater for the greyscale difference map. During the initial erosion stage, the adapted technique calculates the mean of the object cluster block and any pixel values that fall below the mean are set to zero while the remaining pixels are set to one. The resulting binary image is then operated upon during the dilation stage. This concept is based on a similar one described in [Weeks96].

The final stage of the filtering algorithm is to bound the filtered object within a further rectangular block. Ideally the second rectangular block is smaller than the original as it now only encloses valid object data. The remainder of the original object cluster block is assumed to be unwanted background scenery and is thus discarded. In order to bound the object, the algorithm attempts to locate a solid mass of non-zero pixels within the filtered cluster block. If located, it then searches for four definite edges of the object. The bounding rectangle is then drawn around the located edges. If a solid mass of non-zero pixels is not located, then the algorithm assumes the filtered results originated from noise and are thus discarded.

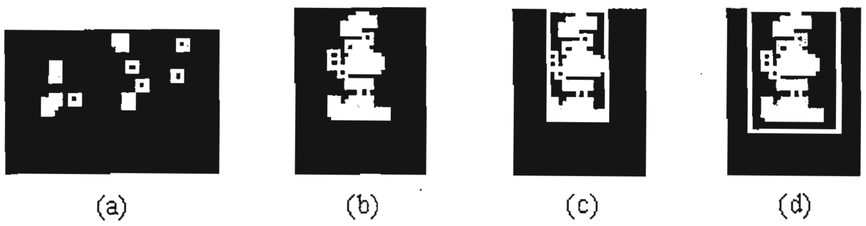


Figure 4-18: Results after morphological opening filtering.

Preliminary morphological filtering test results are illustrated in Figure 4-18. Figure 4-18(a) depicts the result of filtered random noise while Figure 4-18(b) depicts the filtered result of a valid vehicle in motion. Note the shape of the object is clearly distinguishable compared to the noise. Figure 4-18(c) illustrates how the implemented algorithm located the four edges of the object and thus bounded it within a smaller rectangle. The algorithm determined that the result illustrated in Figure 4-18(a) originated from noise and thus it was discarded. As far as encoding is concerned, a refined object needs to be bounded in a rectangle that comprises 8x8 blocks. In the event that the object is not bounded by a correctly sized rectangle, the rectangle's size is increased accordingly until the criterion is satisfied. An example of a resized bounding block is illustrated in Figure 4-18(d). The resulting rectangle co-ordinates as well as the original object cluster block co-ordinates are recorded in the global **Cluster_Coords** buffer. The refined co-ordinates are eventually mapped to the buffered target video frame and the object data is extracted and encoded.

4.6 Object Tracking

The object tracking algorithm was written in C programming language and comprises a main controlling function and a set of callable functions similarly to the segmentation algorithm. The controlling function is called following the object segmentation algorithm. The development of the tracking algorithm followed a similar pattern to the segmentation algorithm in that the code was written and debugged in the BORLAND C++ Builder ver 3.0 SDE before being compiled and uploaded to the TriMedia for further testing.

Table 4-8: Passed object tracking variables.

Passed variables		
int	**mean_diff**	Threshold used for the matching of halted objects within the video scene
int	**ref_matrix_size**	Limits the number of remaining frequency coefficients after applying the DCT
uchar	**track_boundary_top**	Top co-ordinate of user-defined area wherein objects are segmented and tracked
uchar	**track_boundary_bottom**	Bottom co-ordinate of user-defined area wherein objects are segmented and tracked
uchar	**track_boundary_left**	Left co-ordinate of user-defined area wherein objects are segmented and tracked
uchar	**track_boundary_right**	Right co-ordinate of user-defined area wherein objects are segmented and tracked

The GUI passes six user-defined variables pertaining to the tracking of objects to the controlling function. These passed variables are listed in Table 4-8. The object tracking algorithm utilises the same global variables that were declared for and used by the reference frame encoder as well as the object segmentation algorithm.

The following sections describe the implementation of the object tracking algorithm.

4.6.1 Initial Tracking Iteration

The first object tracking iteration is executed in a slightly different manner compared to subsequent executions. The flowchart of the initial tracking iteration is illustrated in Figure 4-19. Assuming the preceding object segmentation algorithm resulted in the segmentation of objects in motion within the video scene, the first step of the tracking algorithm is the allocation of a dynamic data memory structure for each segmented object. The structure template that is declared for each segmented object is listed in Table 4-9. The data structures are organised in a link-list formation, i.e. each structure contains a pointer that points to the address of the next structure within the link-list. Thus the simple execution of a loop allows for the accessing of each buffered object at any given time.

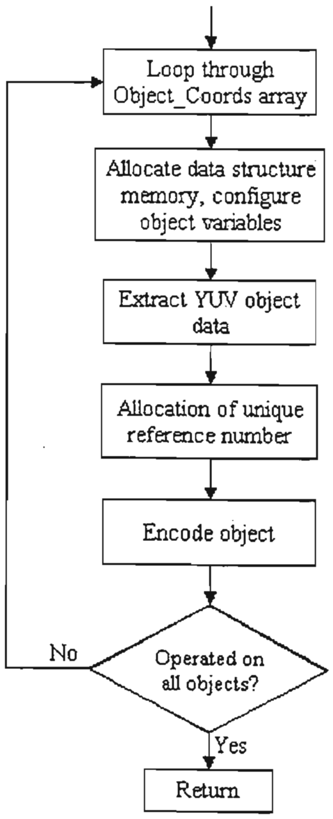

Loop through Object_Coords array
Allocate data structure memory, configure object variables
Extract YUV object data
Allocation of unique reference number
Encode object
Operated on all objects?
No
Yes
Return

Figure 4-19: Initial object tracking algorithm.

Following the structure allocation, the YUV data belonging to each segmented object is extracted from the buffered target video frame and stored within its respective data structure. Each object's co-ordinate arrays are initialised and their **object_TX** flags are set to one. This flag is used to indicate whether an object requires full DCT encoding, or if its calculated motion vector requires encoding. In the case of the initial tracking iteration, each object is newly segmented and thus their extracted YUV components require full DCT encoding. If the flag was set to two for example, then only the motion vector of the object would be encoded.

Each newly buffered object is allocated a unique reference **Object_Number** that is used for identification purposes by both the encoder and decoder. The final step is that of encoding. The encoding process employed is similar to that applied to a reference frame but with the main differences being that an object is much smaller in size and the Y component of the object comprises macroblocks that are fixed in size at 8x8 pixels. The actual encoding process involves the application of the 2D DCT, quantisation and finally RLE.

Table 4-9: Dynamic data structure template used to buffer segmented objects.

Dynamic data structure template		
int	**Object_Number**	Unique reference number used for identification purposes by both encoder and decoder
uchar	***Yarray**	Pointer to dynamic array used to buffer Y component of segmented object
uchar	***Uarray**	Pointer to dynamic array used to buffer U component of segmented object
uchar	***Varray**	Pointer to dynamic array used to buffer V component of segmented object
int	**prev_coords[4]**	Array used to store co-ordinates of bounding rectangle of object in target video frame (n-1)
int	**curr_coords[4]**	Array used to store co-ordinates of bounding rectangle of object in current target video frame (n)
int	**cluster_coords[4]**	Array used to store cluster block co-ordinates of object in current target video frame
int	**motion_vector[2]**	Array used to store motion vector of moving object calculated from target video frames n & n-1
int	**Width**	Bounding rectangle width of refined segmented object
int	**Height**	Bounding rectangle height of refined segmented object
float	**Mean**	Y component mean of bounded object
int	**mem_size**	Indicates the size of dynamic memory allocated for ***Yarray**, for possible reallocation purposes
char	**Object_TX**	Flag indicates if YUV components or motion vector of segmented object are to be encoded
int	**Iteration_count**	Counter indicates the number of successive iterations that a tracked object has not been tracked
struct	***nextPTR**	Address pointer to the next data structure within the link-list

The U and V components of an object are encoded in a different manner to that of the Y component. Considering an object is sub-divided into 8x8 blocks and that the YUV components of an object are 4:2:0 sub-sampled, therefore each Y 8x8 macroblock corresponds to U and V macroblocks that are 4x4 pixels in size. As the implemented 2D DCT function was designed to transform macroblocks with a minimum size of 8x8 pixels, the U and V macroblocks are therefore not transformed or quantised but simply run-length encoded. Once an object has been encoded, the next object in the link-list is encoded and so this continues until all are encoded.

Figure 4-20 provides an example of the composition of the arrays that temporarily buffer the encoded YUV data of the objects. In the illustration, x represents each objects encoded data. Each newly segmented object is preceded by header information required for decoding and display purposes. With regard to Figure 4-20, the header variables are as follows:

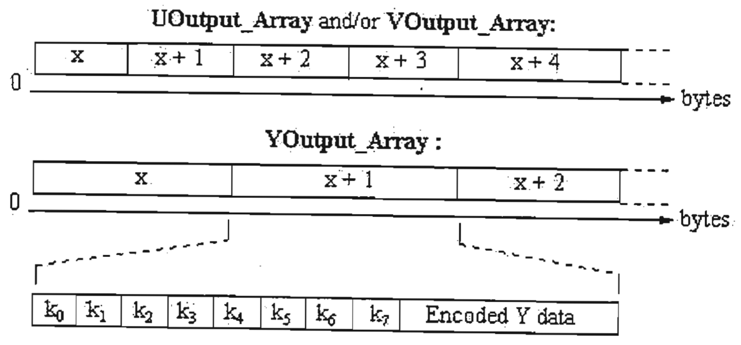


Figure 4-20: Typical composition of encoded object arrays.

k_0 – Object reference number
k_1 – **object_TX** flag
k_2 – height
k_3 – width
k_4 – top left corner of object rectangle row / 256 (offset with regard to top left corner of reference frame)
k_5 – top left corner of object rectangle row % 256 (% is the mathematical modulus)
k_6 – top left corner of object rectangle column / 256
k_7 – top left corner of object rectangle column % 256

The contents of the temporary arrays are then Arithmetically encoded and finally transferred to the decoder.

4.6.2 Successive Tracking Iterations

Assuming the initial tracking iteration resulted in the segmentation and buffering of at least one moving object, the motion of the buffered object(s) is tracked within the subsequent processing iterations. The implemented tracking algorithm is reasonably complex and is thus best described by a series of flowcharts. With regard to the following flowcharts, the difference between a buffered object and a newly segmented object must be clarified. A buffered object is one that was segmented in a past processing iteration while a newly segmented object is one that has been segmented within the current iteration. The tracking of a buffered object involves matching it to its newly segmented counterpart but in an offset position. As a result, motion vectors indicating positional offsets of tracked objects are calculated and transmitted to the decoder.

The operation of the controlling tracking function and the subsequent called functions are portrayed by the flowcharts in Figures 4-21, 4-22, 4-23 and 4-24. Within each flowchart, certain process blocks or decision diamonds are bounded within a labelled dashed rectangle. For example, a dashed rectangle might have the label **COMMENT A**. These rectangles represent important steps within the respective functions and thus warrant further discussion. The discussions accompany the respective flowcharts.

The tracking algorithm makes use of two important variables that indicate the state of each tracked object. The purpose of these two variables is described as follows:

- **iteration_count:** This counter indicates the number of tracking iterations that have passed since a buffered object was last successfully tracked. The terms successfully tracked or a successful tracking iteration indicate the event in which a buffered object has been correctly matched to its newly segmented counterpart within the current processing iteration. Following each successful tracking iteration, this counter is set to zero. On the other hand, the counter is incremented with each consecutive processing iteration that a buffered object is not tracked. If the number of counts exceeds two, it is then assumed that the object is no longer present within the video frame and is thus deleted from memory.
- **object_TX:** This flag indicates the encoding state of an object in each tracking iteration. When the flag has a value of one, an object's YUV components are fully encoded, i.e. DCT, quantisation, RLE etc. If the flag has a value of two, it indicates that the decoder has a copy of the

current version of the buffered object's YUV components thus only its calculated motion vector is encoded.

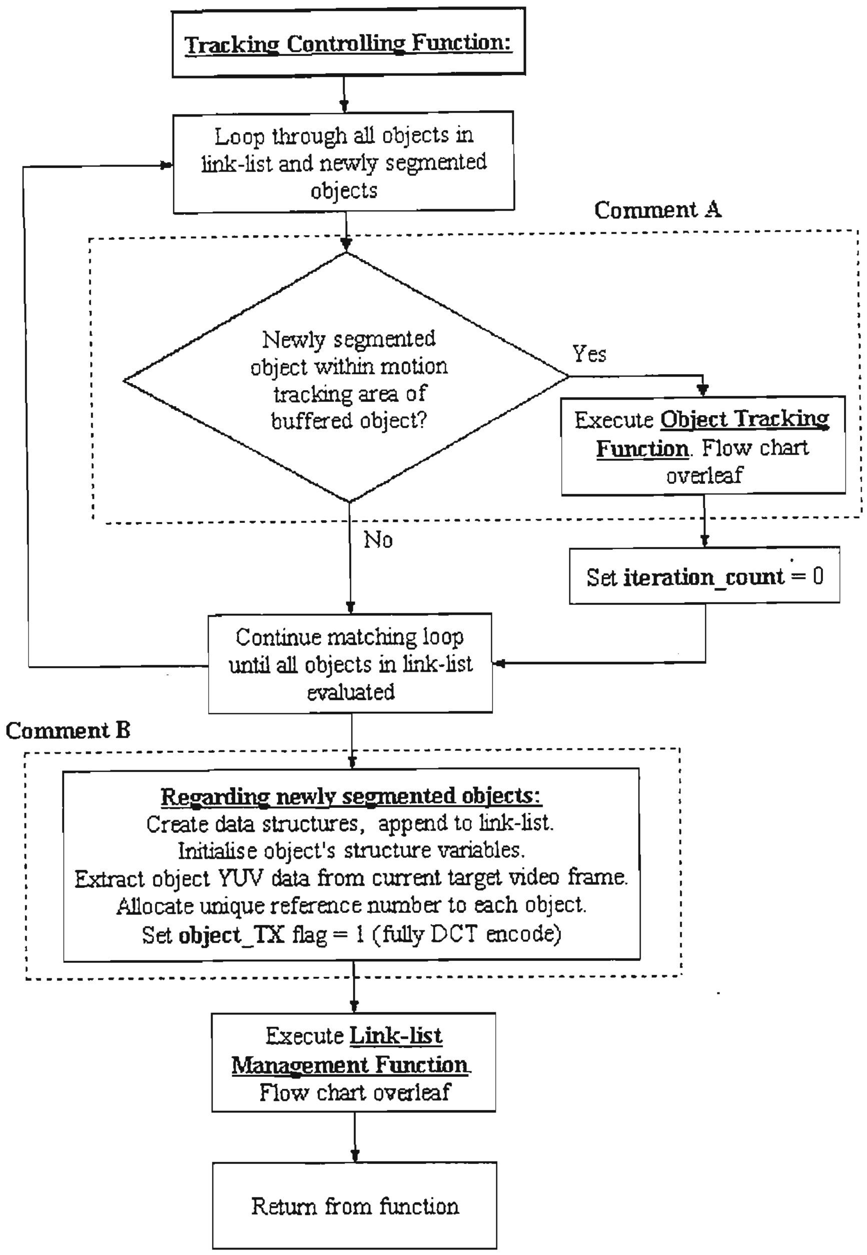

Tracking Controlling Function:
Loop through all objects in link-list and newly segmented objects
Comment A
Newly segmented object within motion tracking area of buffered object?
Yes
Execute Object Tracking Function. Flow chart overleaf
No
Set iteration_count = 0
Continue matching loop until all objects in link-list evaluated
Comment B
Regarding newly segmented objects:
Create data structures, append to link-list.
Initialise object's structure variables.
Extract object YUV data from current target video frame.
Allocate unique reference number to each object.
Set object_TX flag = 1 (fully DCT encode)
Execute Link-list Management Function. Flow chart overleaf
Return from function

Figure 4-21: Tracking controlling function.

COMMENT A:

This comment deals with the Tracking Controlling Function. The top and bottom row co-ordinates and the left and right column co-ordinates of each buffered object within the link-list are compared to those co-ordinates of each newly segmented object. If all four co-ordinate points of a newly segmented object fall within a buffered object's motion tracking area, then it is assumed that the two independent objects are actually the same object, displaced only in time. If this 'match' occurs, the Object Tracking Function is then executed (Figure 4-22). If no match occurs, then the next buffered object within the link-list is selected and the comparison with the newly segmented objects occurs over. This matching process continues until every buffered object within the link-list has been compared to each of the newly segmented objects. The process ensures that a newly segmented object can only be matched to one buffered object and vice versa.

The constraints and limitations of the defined motion tracking area are analysed in the following section (4.6.3).

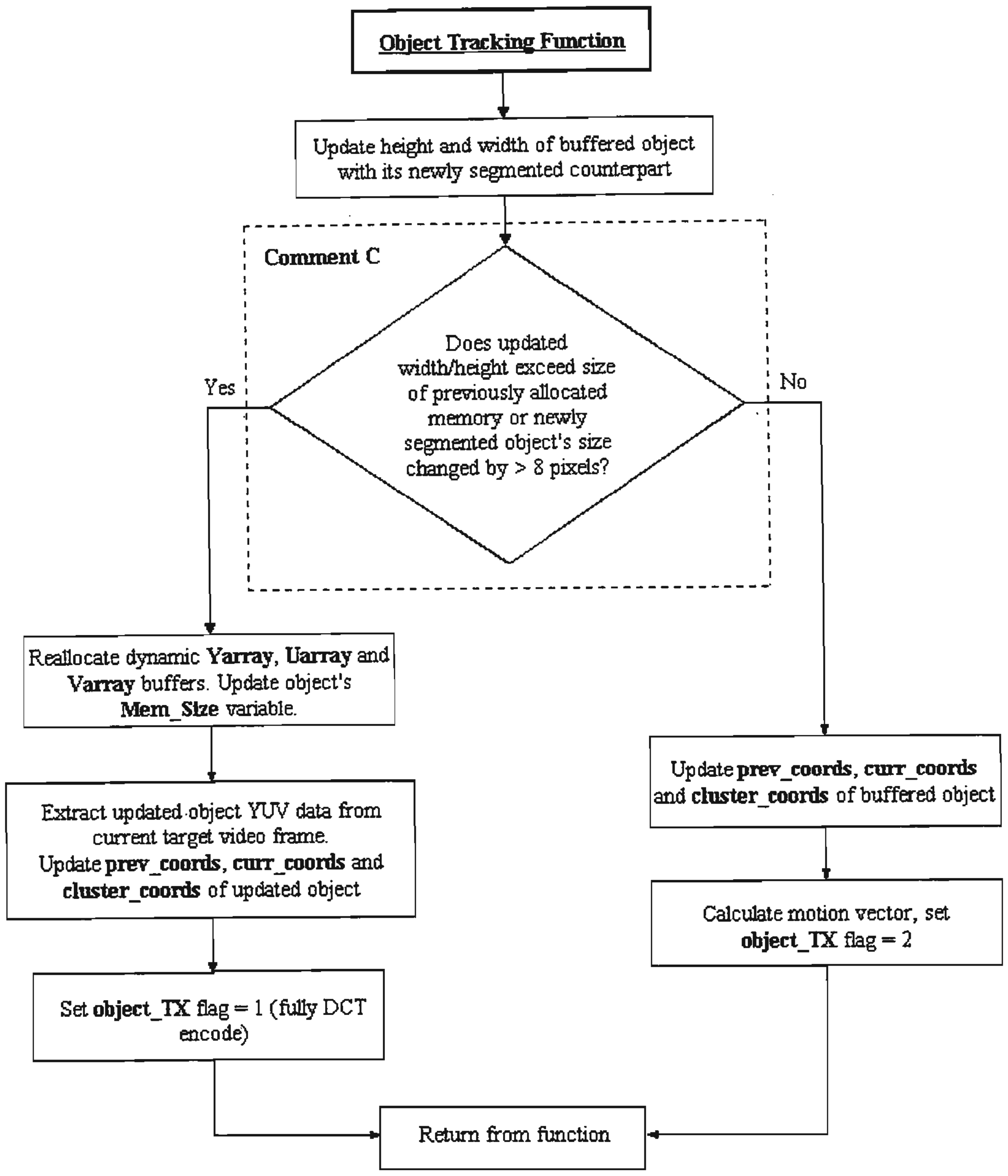

Object Tracking Function
Update height and width of buffered object with its newly segmented counterpart
Comment C
Does updated width/height exceed size of previously allocated memory or newly segmented object's size changed by > 8 pixels?
Yes
No
Reallocate dynamic Yarray, Uarray and Varray buffers. Update object's Mem_Size variable.
Extract updated object YUV data from current target video frame. Update prev_coords, curr_coords and cluster_coords of updated object
Set object_TX flag = 1 (fully DCT encode)
Update prev_coords, curr_coords and cluster_coords of buffered object
Calculate motion vector, set object_TX flag = 2
Return from function

Figure 4-22: Object tracking function.

COMMENT B:
This comment deals with the Tracking Controlling Function. Having matched as many buffered objects within the link-list to newly segmented objects as possible, the remaining unmatched newly segmented objects are allocated their own dynamic data structures that are appended to the link-list as it is assumed that they constitute newly detected objects in motion within the video scene. The YUV components of the segmented objects are extracted from the target video frame and buffered accordingly.

COMMENT C:
This comment deals with the Object Tracking Function. It is possible that the size of a newly segmented object differs from that of its buffered counterpart due to the object moving in or out of the video scene, rotation, zooming or occlusion. If this is the case, the sizes of the dynamic arrays are then

increased by means of reallocation and the YUV components of buffered object are updated by extracting them from the current target video frame. If the object has not changed size, then the co-ordinates of the buffered object are updated and the motion vector corresponding to the difference between it's previous and current position within the video scene is calculated. In this case, the **object_TX** flag is set to two indicating that only the motion vector of this particular object requires encoding.

COMMENT D:
This comment deals with the Link-list Management Function. The purpose of this step is to analyse the tracking status of each buffered object. Newly segmented objects as well as buffered objects that have been successfully tracked within the current tracking iteration have their **iteration_counter**s set to zero. The **iteration_count** variable is incremented on the other hand if a buffered object is not successfully tracked. In the situation where an object has not been tracked within three consecutive tracking iterations, a search within the current target video frame is executed in an attempt to locate the object (refer to **COMMENT E**). Possible reasons for the object not being tracked within three consecutive tracking iterations include the object moving out of the video scene, occlusion as well as the object originating from unfiltered noise.

COMMENT E:
This comment deals with the Link-list Management Function. In the situation where a buffered object has not been tracked for three consecutive tracking iterations, a diamond zonal search within the current target video frame is executed. Its purpose is to determine if the object is still visible within the video frame. The search is conducted by calculating the SAD values of the Y component of the buffered object and its possible counterparts within the target video frame within a diamond zonal region. The search is executed by starting at the object's last segmented position. Assuming that a best match has been located, as a final test for similarity the mean values of the two objects are calculated and compared. If the difference falls below the user defined **Mean_diff** threshold, then it is assumed that a suitable match for the buffered object has been located and therefore the object is still visible within the current frame. As a result, the buffered co-ordinates of the object are updated and its motion vector is calculated. Its **iteration_count** is set to zero. If no matching object within the current target video frame is located, then it is assumed that the object is no longer present (visible) and is thus deleted from the link-list. The resultant effects of varying **Mean_diff** are presented in Chapter 5.

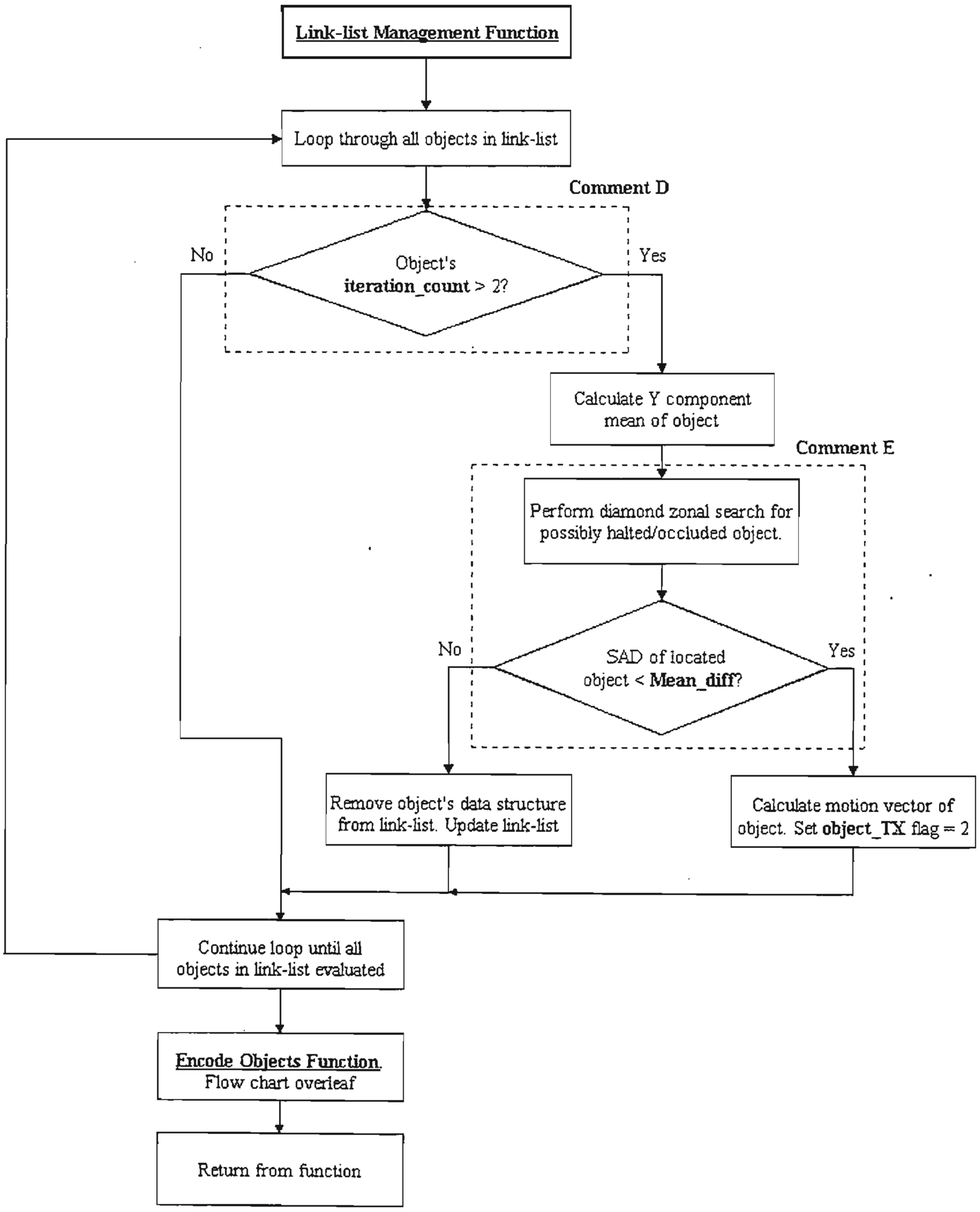

Link-list Management Function
Loop through all objects in link-list
Comment D
No
Object's iteration_count > 2?
Yes
Calculate Y component mean of object
Comment E
Perform diamond zonal search for possibly halted/occluded object.
No
SAD of located object < Mean_diff?
Yes
Remove object's data structure from link-list. Update link-list
Calculate motion vector of object. Set object_TX flag = 2
Continue loop until all objects in link-list evaluated
Encode Objects Function. Flow chart overleaf
Return from function

Figure 4-23: Link-list management function.

COMMENT F:

This comment deals with the Encode Objects Function. It is possible that as a result of each tracking iteration:

- a buffered object could have been successfully tracked,
- been unsuccessfully tracked
- been deleted from the link-list

- changed shape
- or that a newly segmented object may-have been added to the link-list

In the last two situations, the YUV components of the buffered object are new and are thus fully encoded by application of the implemented 2D DCT, quantiser and RLE functions. The encoding process follows the sequence described in Section 4.6.1. In the event that an object has been successfully or unsuccessfully tracked, i.e. the object still exists within the link-list and its **iteration_count** is less than three, the following data pertaining to the object is encoded:

- its reference number
- **object_TX** flag,
- vertical motion vector
- horizontal motion vector

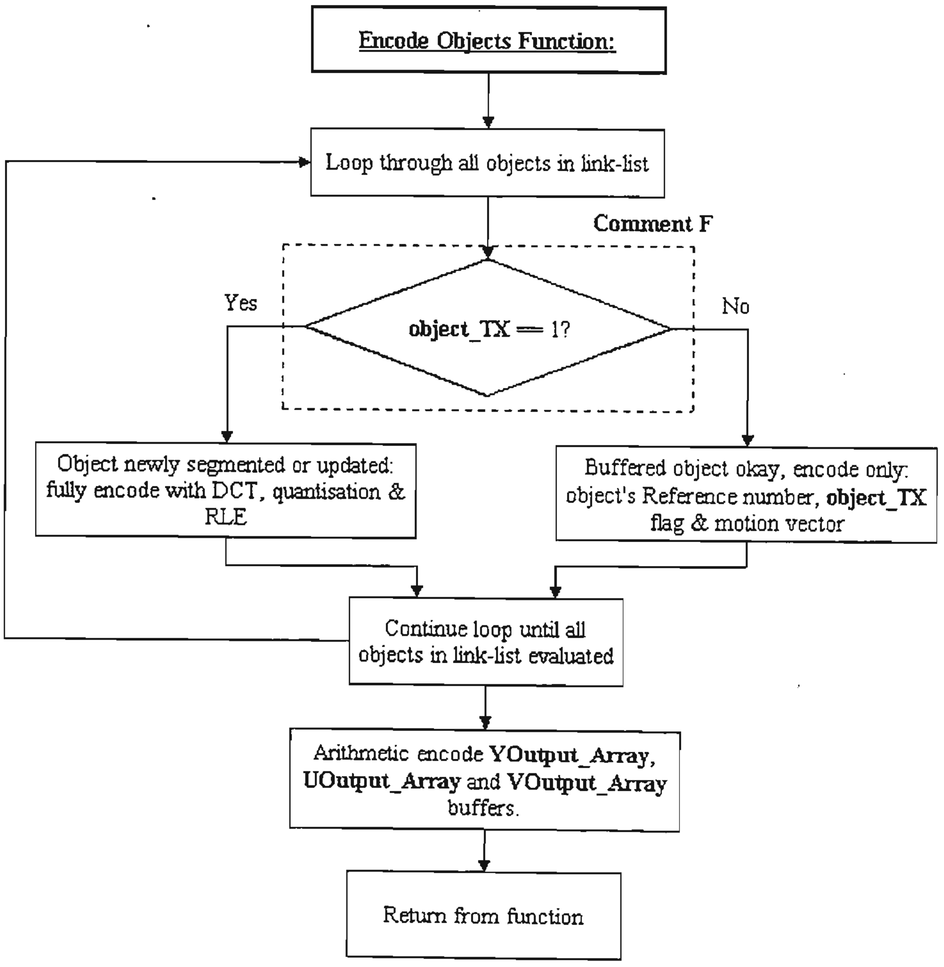

Encode Objects Function:
Loop through all objects in link-list
Comment F
Yes
No
object_TX == 1?
Object newly segmented or updated: fully encode with DCT, quantisation & RLE
Buffered object okay, encode only: object's Reference number, object_TX flag & motion vector
Continue loop until all objects in link-list evaluated
Arithmetic encode YOutput_Array, UOutput_Array and VOutput_Array buffers.
Return from function

Figure 4-24: Encode objects function.

The run-length encoded components of all of the objects within the link-list are then combined, Arithmetically encoded and then transferred to the decoder. As described previously, if an object's **iteration_count** exceeds two, then the object is deleted from memory at both the encoder and decoder.

4.6.3 Analysis of the Motion Tracking Area

Recall from Chapter 3 that in order for a buffered object to be successfully tracked within a current processing iteration, its newly segmented counterpart must fall within the buffered object's defined motion tracking area. As the motion tracking area is a fixed region surrounding an object, it therefore has a direct impact on the speed at which an object can travel if it is to be tracked between successive processing iterations. A further influence in terms of tracking is the focal length of the video camera. Naturally if the camera has a long focal length and thus an object within the scene is small, then it can be safely assumed that the object can travel at a greater speed than what it could if the focal length was shorter hence making the object appear bigger, if it was still to be successfully tracked.

In order to analyse this type of situation, we use an example in association with (4-7). Note that the delay between the capturing of five frames is timed from the point at which Frame 0 is captured to the time at which Frame 4 is captured which effectively corresponds to 160 milliseconds. Assume an average vehicle is 4.5 meters long and 1.8 meters wide. Looking at the vehicle from a directly overhead position (top view), if the vehicle is represented in a video scene by a rectangular block 10 pixels wide and 25 pixels long, each pixel within the scene therefore corresponds to 0.18 meters. If the vehicle is travelling at 20 km/h, using (4-7) where f is the speed in km/h and g is the meters represented by each pixel in the video scene (in this case 0.18 meters) and the motion is detected over five frames hence t is 160 milliseconds, the number of pixels the vehicle moves between consecutive processing iterations is approximately five. On the other hand, if the vehicle is travelling at 100 km/h, the number of pixels the vehicle moves between iterations is approximately 24.7.

$$pixels = \frac{\left(\frac{f}{3.6}\right).t}{g} \tag{4-7}$$

If the video camera's focal length was shortened and the same vehicle was now represented by a block 30 pixels wide and 75 pixels long thus each pixel corresponds to 0.06 meters, the number of pixels the vehicle would move between iterations if travelling at 20 km/h would be approximately fifteen. If the vehicle were travelling at 100 km/h, it would move 74 pixels between consecutive iterations. Figure 4-25 provides a graphical analysis of (4-7) where the two variables f and g are independently adjusted while the resulting minimum number of pixels that a vehicle must move between processing iterations in order to be tracked is given by the vertical axis.

Technically speaking, the defined motion tracking area, given by x in Figure 4-26, of an object is content specific. Based on the analysis in Figure 4-25, if a particular scene was filmed using a video camera with a long focal length, a motion tracking area of 20 (pixels) or above should prove more than adequate if wanting to track objects that travel at speeds up to 100km/h. On the other hand, if the camera had a shorter focal length, a larger motion tracking area would have to be defined if wanting to track the object at reasonably high speeds. Ideally the motion tracking area should be a variable that would be defined by the system operator in order to configure the tracking capability of the system depending on the type of scene at hand. However in order to determine a suitable motion tracking area for evaluation and demonstration purposes, a general vehicle scenario was considered. The calculated tracking area was based on the assumption that we wish to track vehicles within a suburban area and thus their maximum speeds should be 60 km/h. A further assumption made was that a reasonable size of vehicles to be tracked, compared proportionally to the size of the video scene, is illustrated in Figure 4-27.

In Figure 4-27, the width of the objects is approximately sixteen pixels and based on the previous approximation that a vehicle is 1.8 meters wide, therefore each pixel within the scene represents 0.11 meters. Applying these values to (4-7), the utilised motion tracking area was therefore calculated at approximately 24 pixels. Once again this value is not necessarily suitable to all types of video scenes. It was however felt that reasonable conclusions could be drawn from this utilised value based on the results obtained and presented in Chapter 5.

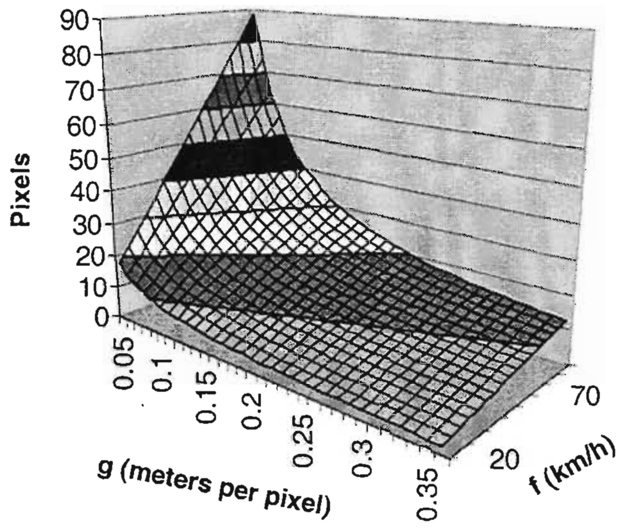


Figure 4-25: Graphical analysis of Equation 4-7.

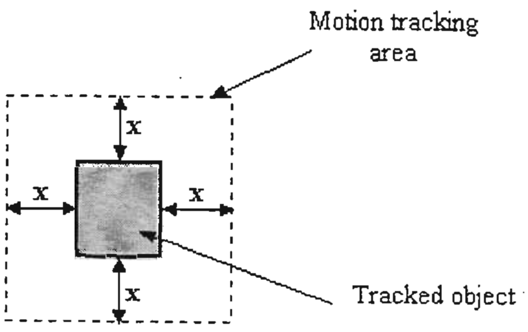


Figure 4-26: Boundary limitations of an object's motion tracking area.

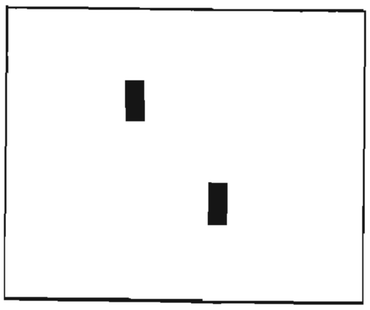


Figure 4-27: Ideal size of objects to be tracked compared proportionally to the size of a video scene.

4.6.4 *Preliminary Testing*

Testing the implemented object segmentation and tracking algorithms within the BORLAND C++ Builder software environment identified potential shortcomings of the system in terms of its consistency in filtering ambient noise. For example, certain test video scenes in which the video camera had a long focal length, digitisation jitter concentrated around detailed objects within the scene featured prominently. The net result of the unfiltered noise was erroneous object segmentation and tracking. An example of such a situation is illustrated in Figure 4-28 where (a) illustrates a particular target video frame and (b) illustrates the segmentation results.

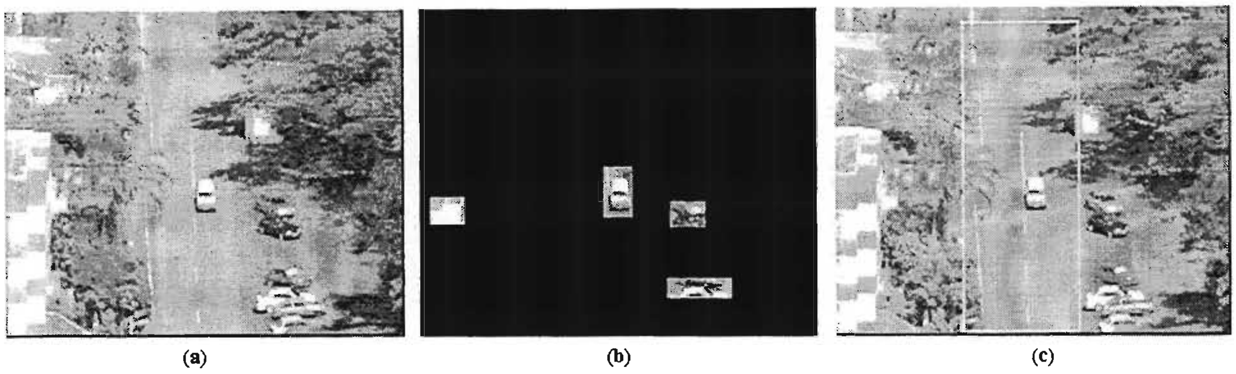


Figure 4-28: Implemented user-defined boundary tracking region.

It was reasoned that in a video scene, there is generally a specific area of interest in which a system operator's attention will be focussed. In Figure 4-28(a) for example, the area of interest is the road in the centre of the scene as it is within this area that you expect objects to move. Therefore it was decided in order to reduce the likeliness of erroneous segmentation and tracking, the segmentation and tracking area within the video scene would be more focussed by means of the system operator defining an area of interest within the video scene. Thus any noise outside of the defined area would be ignored. An example of a user defined tracking area is illustrated in Figure 4-28(c). In this case, the bounding rectangle ensures that the segmentation and tracking algorithm is concentrated with the road area of the video scene thus reducing the potential negative effects of surrounding noise.

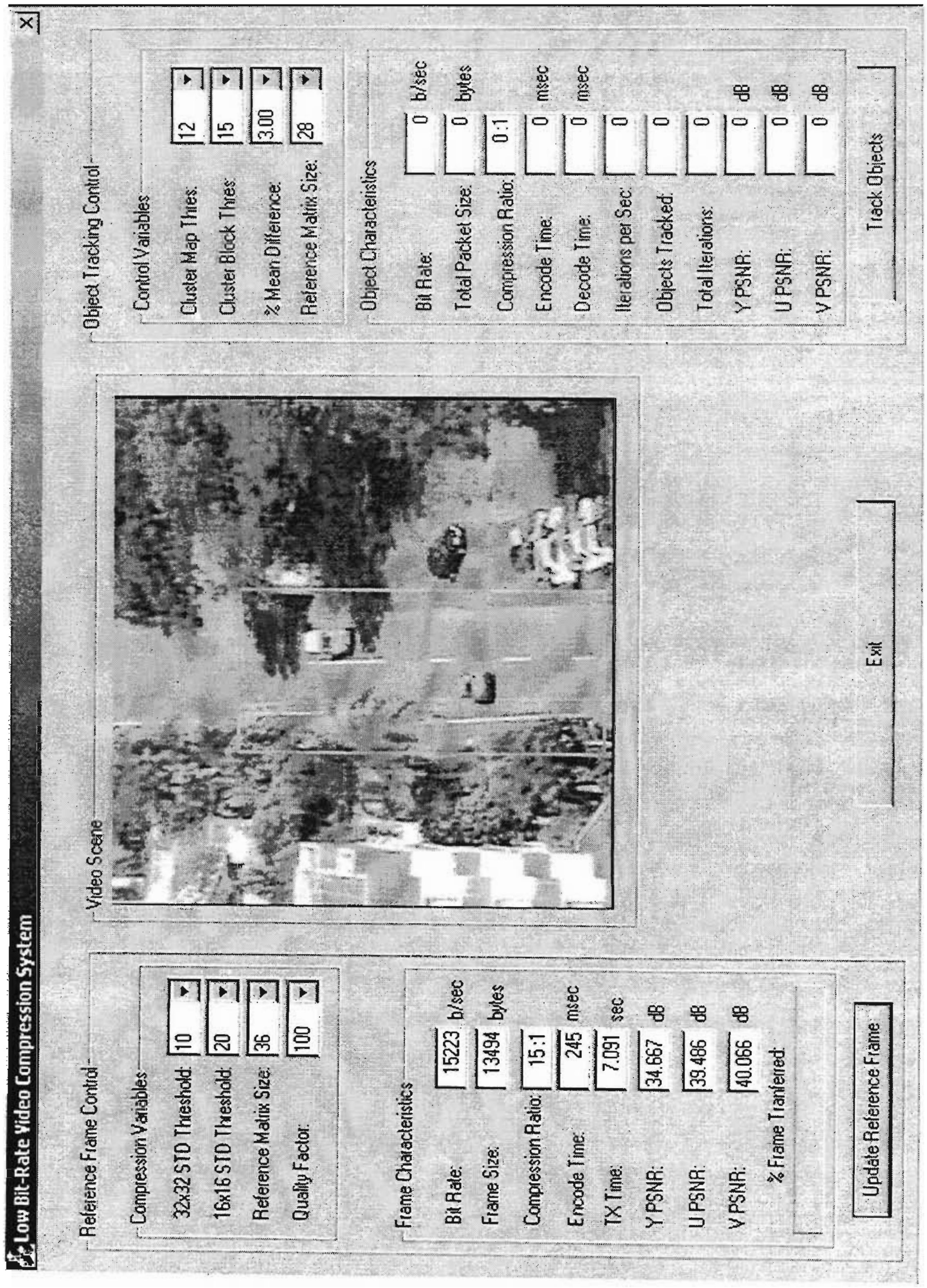
The ability to define an area of interest within a video scene was incorporated into the functionality of the GUI. Its operation is described in greater detail in Section 4.7.3.

4.7 Graphic User Interface

The GUI was written in Microsoft Visual C++ (MSVC++) version 6.0, professional edition. Low level testing and debugging of the GUI was performed by stepping through the code and monitoring respective variables within variable watch windows. The GUI was designed in order to allow the system operator to:

- configure respective compression variables
- view calculated rate metrics
- view the decoded video stream

all within a graphic interface displayed on the PC's monitor. Figure 4-29 is a screen capture of the implemented GUI .


Low Bit-Rate Video Compression System

Reference Frame Control

Compression Variables

32x32 STD Threshold: 10
16x16 STD Threshold: 20
Reference Matrix Size: 36
Quality Factor: 100

Frame Characteristics

Bit Rate: 15223 b/sec
Frame Size: 13494 bytes
Compression Ratio: 15:1
Encode Time: 245 msec
TX Time: 7.091 sec
Y PSNR: 34.667 dB
U PSNR: 39.486 dB
V PSNR: 40.066 dB
% Frame Tranferred:

Video Scene

Object Tracking Control

Control Variables

Cluster Map Thres: 12
Cluster Block Thres: 15
% Mean Difference: 3.00
Reference Matrix Size: 28

Object Characteristics

Bit Rate: 0 b/sec
Total Packet Size: 0 bytes
Compression Ratio: 0:1
Encode Time: 0 msec
Decode Time: 0 msec
Iterations per Sec: 0
Objects Tracked: 0
Total Iterations: 0
Y PSNR: 0 dB
U PSNR: 0 dB
V PSNR: 0 dB

Figure 4-29: Screen capture of the implemented GUI.

The GUI is divided into three main sections, namely the reference frame control, object tracking control and the decoded stream display area. The following sections describe the general functionality of the implemented GUI as well as providing a more in-depth discussion of the reference frame and object tracking controls.

4.7.1 Structure and Operation

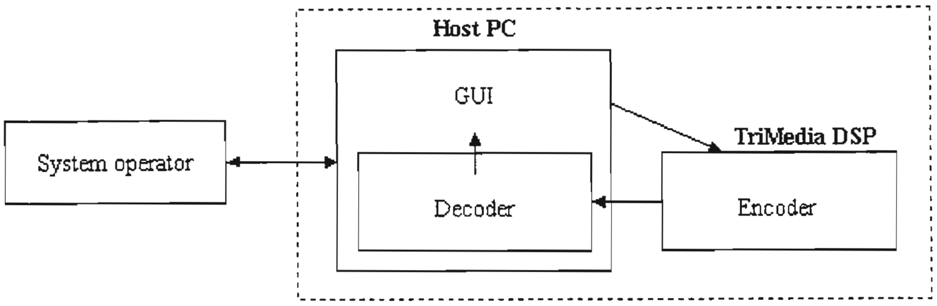


Figure 4-30: GUI relationship.

The relationship of the GUI with respect to the system operator, encoder and decoder is illustrated in Figure 4-30. The GUI enables the system operator to configure video compression variables as well as control the operation of the compression system by means of various control buttons. Following their configuration, the variables are passed to the encoder. Encoded frame or object data is then transferred to the decoder along with corresponding rate metrics. The decoder then decodes the received data and the results are presented to the system operator through the GUI. This cyclical process continues over and is directly controlled by the system operator.

The Microsoft Foundation Class (MFC), which is the main class central to any dialog-based application created within MSVC++, distinguishes between two types of threads: *UI threads* that generally perform user interface tasks, and *worker threads* that are generally used to perform operations 'behind the scene' [InformIT01]. The GUI was designed such that on start-up, two threads are created. The first thread is a UI thread whose primary task is to control and monitor the user interface. The second created thread is a worker thread whose tasks include controlling the transfer of data between the GUI, decoder and TriMedia as well as performing all decoding operations. Considering the two threads operate independently and that by default they are allocated their own stacks within the PC's data memory area [Sams97], a mechanism had to be introduced to control the sharing of common data between the threads. This was achieved by implementing common global variables to which both threads would have read/write access and with flags serving as semaphores used to synchronise their usage.

A problem that was encountered when dealing with two different types of threads is that certain member functions within the UI thread are not accessible by the worker thread. This is essentially due to the two threads inheriting from different classes in the hierarchical tree and obviously the different classes contain different member functions. The inaccessibility problem posed was that after the worker thread had decoded a video frame for example, it would be unable to directly update the GUI with the video frame and rate metrics. Although the use of global variables is a partial solution to this problem, the UI thread cannot sit in a continuous loop monitoring the global variables and their control flags. This is because the UI thread is actually event driven whose primary task is to perform an action when an event is triggered, for example when a button is pressed on the UI. The problem was solved by means of a system timer. The timer acts as a link between the two threads as upon triggering every 200 milliseconds, data could be transferred between the two threads, by means of the global memory space, in either direction depending on the status of the respective control flags (semaphores). This concept is illustrated in Figure 4-31.

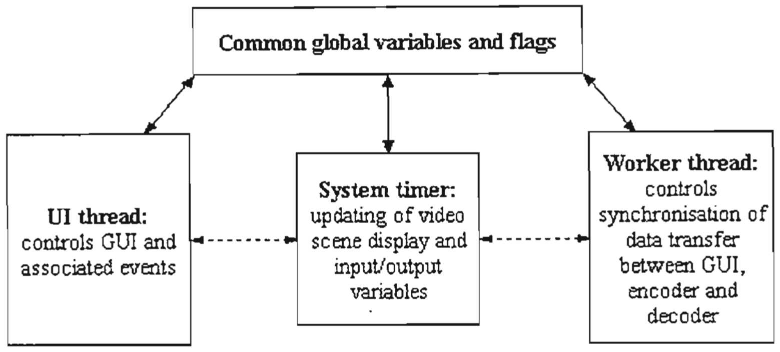

Common global variables and flags
UI thread: controls GUI and associated events
System timer: updating of video scene display and input/output variables
Worker thread: controls synchronisation of data transfer between GUI, encoder and decoder

Figure 4-31: Functional purpose of the implemented system timer.

At the conclusion of an encoding iteration, accompanying the transfer of encoded video data to the decoder is the original unencoded target video frame in YUV format. The purpose of this transfer is for the worker thread to calculate the YUV PSNR rate metrics of the decoded video frame with respect to its unencoded counterpart. The transferred target video frame is not considered when calculating the data bit-rate as it is transferred purely for statistical purposes.

A MFC BITMAP API was used to display each decoded video frame. The API requires each frame to be in BITMAP format in RGB colour space. Thus following decoding, each video frame is converted from YUV to RGB colour space before being displayed on the GUI. The worker thread also executes this conversion process. Another important function of the worker thread is the control of the TriMedia. The start up and closing down procedure of the TriMedia follows a very specific synchronised format. In the event that the TriMedia is not closed down correctly, the TriMedia effectively 'hangs' and will not restart unless the PC is reset. Thus the worker thread has been designed to stringently control the starting and closing down TriMedia procedures with error trapping procedures incorporated in order to minimise the likelihood of the TriMedia hanging at any time.

4.7.2 Reference Frame Control

The reference frame control section is located on the left hand side of the GUI. It enables the system operator to configure the four compression variables that are passed to the encoder before the frame encoding process commences. These variables include the two standard deviation variable block size sub-division thresholds (**Thres_32** and **Thres_16**), the reference matrix size (**ref_matrix_size**) variable that determines the number of coefficients that remain after application of the coefficient limiting algorithm and the quantisation quality factor (**Qfactor**). The flowchart in Figure 4-32 illustrates the operation and integration of both the UI and worker threads when a frame is encoded.

The "Update Reference Frame" button is designed such that on depressing, all current operations, if any are currently executing, are stopped. The sequence then executed is as follows:

- All global variables are refreshed
- The current user defined compression variables are read from the GUI
- The variables are then transferred to the encoder

- The encoding process is then executed

The resultant encoded video frame is then transferred to the decoder in data packets with the bit-rate being controlled by the implemented bit-rate regulator. The total transfer time of the encoded reference frame is displayed on the GUI as the "TX Time" variable.

The received video frame is then decoded, upsampled and buffered. The YUV PSNR is calculated with respect to the original target video frame and finally the YUV video frame is converted to RGB colour space ready for display.

The current implementation of the system is limiting in that the reference video frame has to be manually updated by means of the system operator depressing the "Update Reference Frame" button. It is realised that an improved system would automatically update the reference frame periodically. It is however felt that for evaluation and demonstration purposes, the current implementation is sufficient.

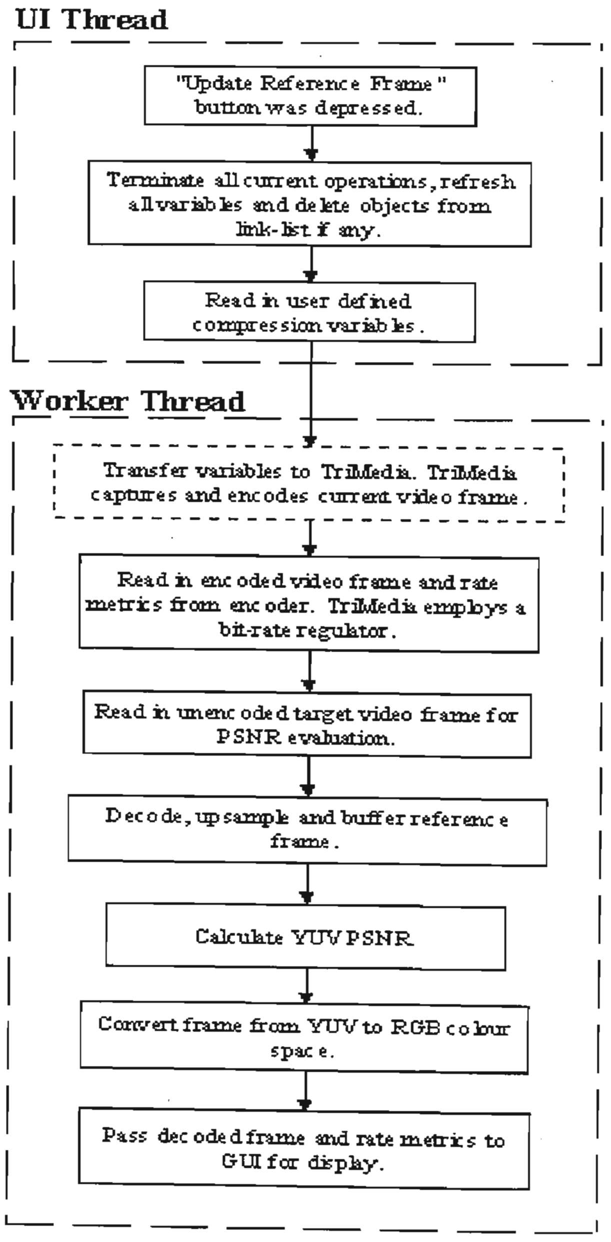

UI Thread
"Update Reference Frame" button was depressed.
Terminate all current operations, refresh all variables and delete objects from link-list if any.
Read in user defined compression variables.
Worker Thread
Transfer variables to TriMedia. TriMedia captures and encodes current video frame.
Read in encoded video frame and rate metrics from encoder. TriMedia employs a bit-rate regulator.
Read in unencoded target video frame for PSNR evaluation.
Decode, upsample and buffer reference frame.
Calculate YUV PSNR
Convert frame from YUV to RGB colour space.
Pass decoded frame and rate metrics to GUI for display.

Figure 4-32: Reference frame encoding flowchart.

4.7.3 *Object Tracking Control*

The object tracking control section, located on the right hand side of the GUI, enables the system operator to configure the four control variables that are passed to the TriMedia encoder before object tracking commences. These variables include the Y cluster map threshold (**ClusMap_Thres**), the segmented cluster block mean threshold (**ClusBlock_Thres**), the percentage mean difference threshold used during the locating of a possibly halted tracked object (**Mean_diff**) as well as the coefficient limiting reference matrix size (**ref_matrix_size**) variable. The flowchart in Figure 4-33 illustrates the integration and operation of the UI and worker threads.

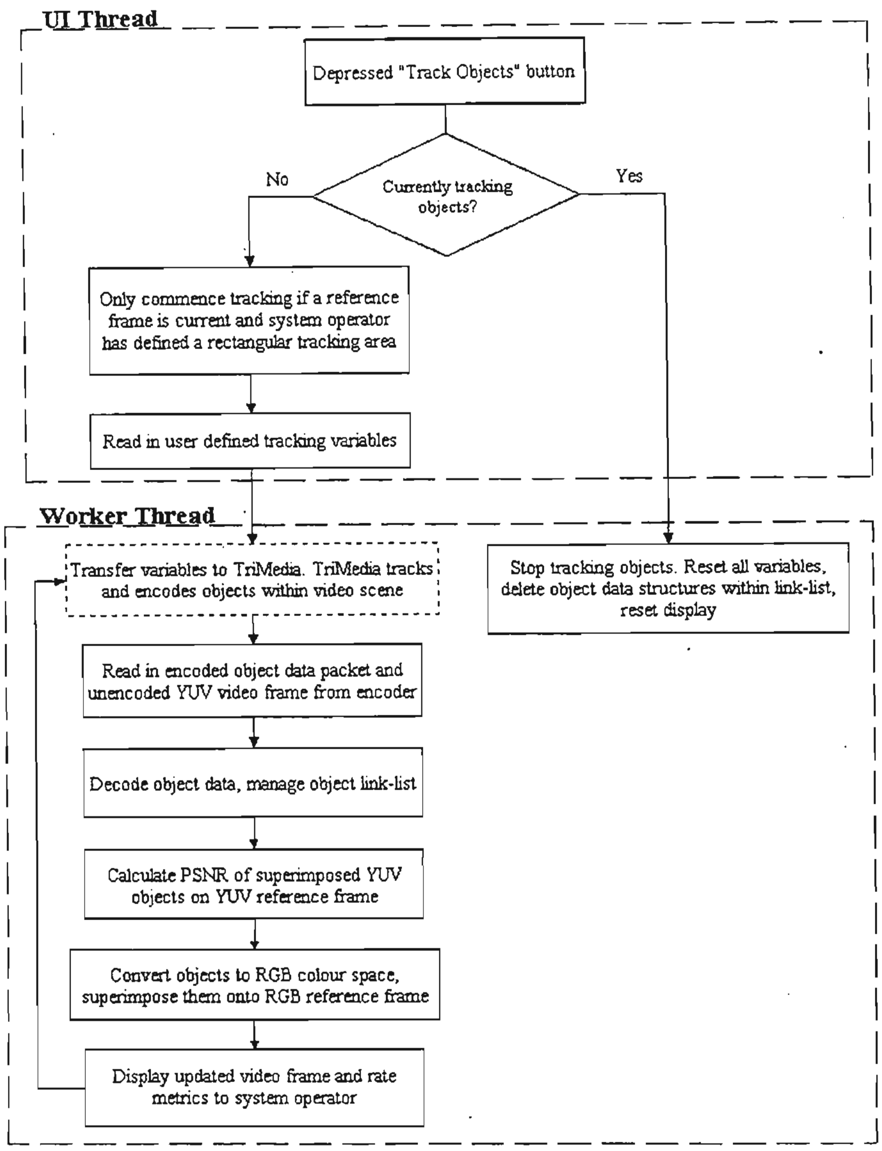

UI Thread
Depressed "Track Objects" button
Currently tracking objects?
No
Yes
Only commence tracking if a reference frame is current and system operator has defined a rectangular tracking area
Read in user defined tracking variables
Worker Thread
Transfer variables to TriMedia. TriMedia tracks and encodes objects within video scene
Stop tracking objects. Reset all variables, delete object data structures within link-list, reset display
Read in encoded object data packet and unencoded YUV video frame from encoder
Decode object data, manage object link-list
Calculate PSNR of superimposed YUV objects on YUV reference frame
Convert objects to RGB colour space, superimpose them onto RGB reference frame
Display updated video frame and rate metrics to system operator

Figure 4-33: Object tracking flowchart.

The "Track Objects" button is used as both a start and stop button with regard to the tracking of objects. On depressing the button, the UI thread determines if the tracking process is already executing. If yes, the tracking process is stopped, all tracking variables are refreshed and all currently tracked objects (if any) are deleted. If the tracking algorithm is not already executing, the UI thread then determines if a reference video frame is currently buffered and if the system operator has defined an area of interest within the video scene. The area of interest is defined by left clicking and holding the mouse button down and dragging the mouse pointer from the selected top left corner of the rectangle across the frame towards the bottom right corner. Once the mouse button is released, the area within the rectangle then becomes the area of interest.in which the segmentation and tracking algorithms will be concentrated. The four co-ordinates of the rectangular area are extracted and ultimately transferred to the encoder along with the control variables.

Once a processing iteration has completed, the resulting encoded object data, its size in bytes and the elapsed processing time in milliseconds are transferred to the decoder. After decoding and upsampling of the YUV components of the objects, the worker thread then superimposes the objects onto a copy of the buffered YUV reference frame. The respective PSNR of the created video frame is then calculated with respect to the original YUV target video frame. The buffered objects are then converted to RGB colour space and superimposed onto a copy of the buffered RGB reference frame. The purpose of buffering a copy of the RGB reference frame, although memory intensive, is to speed up the RGB conversion process. This is achieved as instead of converting each created superimposed YUV video frame to RGB colour space, the implemented process only converts the much smaller objects thus reducing processing time.

The created video frame is then displayed to the system operator accompanied by calculated rate metrics that include:

- the instantaneous bit-rate
- compression ratio
- elapsed encoder and decoder processing times
- the resulting number of iterations that are executed per second
- a counter indicating the number of objects that are currently being tracked within the video scene.

The segmentation and tracking process once started, occurs continuously until the "Track Objects" button is depressed once again.

4.8 Summary

This chapter describes both the hardware and software implementation of the low bit-rate video compression system. It begins by describing the general hardware composition of the system in terms of a video camera, encoder, transmission channel, decoder and display. The implemented system makes use of a Philips TriMedia TM1300 DSP card that plugs into a PCI slot of a host PC. As the TriMedia is an independent processor, it was implemented as the proposed system's encoder while the host PC on the other hand was used as the system's decoder, display and user interface. The construction and processing operation of the TriMedia was then described with special mention of the TriMedia's ability to share data across the PCI bus with the host PC.

A description of the implemented reference frame encoder was then presented. It was described how the frame encoder is similar to the JPEG still image compression standard, but with the main difference being that the Y component of the image is divided into variable sized blocks depending on calculated spatial density. Where possible, the encoding functions were optimised in terms of eliminating unnecessary mathematical operations in order to reduce overall processing time. A discussion of the implemented reference frame decoder follows that of the encoder.

The second major function of the encoder is that of object segmentation and tracking. The implemented segmentation process involves the detection of objects in motion within the video scene by means of frame differencing. Improvements were made to the proposed segmentation algorithm in terms of detecting motion within the video scene by differencing over five input frames rather than two consecutive video frames. The reason for this is that after analysis, it was determined that in order to segment reasonable sized vehicles moving at 20 km/h for example, not enough motion information was conveyed within two consecutive video frames thus deterring from accurate object segmentation. Having detected areas of motion within the video scene, the implemented object clustering segmentation algorithm was then described. The segmentation process involves the construction of a cluster map from a Y difference map and the grouping of neighbouring similar valued cluster elements into object cluster blocks. In order to filter ambient noise within the video scene, several filtering stages were incorporated including thresholding and morphological filtering.

The implemented object tracking process was then described. The discussion outlined how the algorithm tracks the *motion* of a segmented object rather than the object itself. The YUV components of newly segmented objects are copied and buffered within dynamically allocated data structures that are organised in a link-list formation. The components of these newly segmented objects are encoded similarly to the reference frame. Within subsequent processing iterations, motion vectors are used to describe the motion of the objects. The tracking algorithm caters for objects that change in size as they possibly move across the video frame border or due to the affects of occlusion. An area of interest within the video scene with regard to object tracking was introduced. The purpose of this user defined area is to reduce potential erroneous object tracking as only segmented objects within the area are tracked and all other objects are regarded as noise and therefore ignored.

The chapter concluded by describing the implemented GUI. The GUI allows the system operator to configure compression variables, control the operation of the encoding and tracking processes as well as view the decoded video stream and corresponding rate metrics. The GUI makes use of two independent threads, one is used to manage the GUI itself while the other is used to control the transfer of data between the encoder and decoder as well as performing all decoding operations.

The following chapter evaluates the performance of the implemented system by discussing the obtained results.

CHAPTER 5

PERFORMANCE EVALUATION

This chapter evaluates the performance of the implemented low bit-rate video compression system. It begins by evaluating the performance of the frame encoder and decoder and this is then followed by an evaluation of the object segmentation and tracking algorithm. The obtained results are discussed both subjectively and objectively.

5.1 Measurement Criteria

Five measurement criteria are used to evaluate the performance of the system. These include:

- **Bits per Pixel (bpp):** measures the ratio between the number of bits of encoded data and the total number of pixels required by its original unencoded counterpart. Bpp is calculated according to:

$$bpp = \frac{x*8}{k} \qquad (5\text{-}1)$$

  where x represents the size, in bytes, of an encoded reference video frame and k represents the total number of pixels per unencoded video frame. Considering the implemented video compression system encodes CIF video in the YUV colour space that is sub-sampled according to the 4:2:0 ratio, k is equal to the total number of Y, U and V pixels within an unencoded video frame totalling (101,376) + 2*(25,344) = 152,064 pixels.

- **Compression Ratio (CR):** defined as the ratio between the size in bytes of an unencoded video frame and its encoded counterpart. Compression ratio is directly related to bpp and can be calculated according to:

$$Compression \quad Ratio = \left(\frac{1}{bpp}\right)*8 \qquad (5\text{-}2)$$

- **Peak Signal to Noise Ratio (PSNR):** ratio measures the logarithmic ratio of the mathematical mean of the squared error between an unencoded input video frame and its decoded counterpart. PSNR is measured in decibels and is calculated on a per frame basis. The resulting PSNR provides an indication of the amount of distortion that is introduced as a frame is encoded. PSNR is calculated according to:

$$PSNR = 10.\log\left[\frac{255^2}{\frac{1}{k}\sum_{i=0}^{k-1}(x_i - \hat{x}_i)^2}\right] dB \qquad (5\text{-}3)$$

  where the maximum pixel value within each of the YUV components is 255, k represents the total number of pixels per colour space, i.e. there are 101,376 pixels in the Y colour space and 25,344 pixels in both the U and V colour spaces, x_i is the value of the i^{th} pixel within an unencoded video frame and $\hat{x}_i$ is the value of the i^{th} pixel in the respective encoded / decoded video frame.

- **Bit-rate:** data transfer rate of the number of bits that pass a given point in a given amount of time, usually a second. In the implemented system, bit-rate is used to measure the amount of data that is transmitted between the encoder and decoder and is measured in bits/second. Bit-rate is calculated according to:

$$bit-rate = \frac{t*8}{u} \tag{5-4}$$

where t is the measured amount of transmitted data in bytes and u is the time difference between the transmission of the first bit of the transmitted data, and the reception of the last bit measured in seconds.

- **Human impression:** this criterion involves the general impression that a human viewer has of a decoded video frame or sequence. Recall that lossy video compression is achieved by discarding image and video information that a human viewer would be visually unable to detect. The greater the amount of image and video information that is discarded, the higher the compression ratio hence the lower the bit-rate of the transmitted video stream. Video compression systems are often designed in order to satisfy a particular bit-rate requirement; in other words, the system has to discard a certain amount of image and video information in order to meet the specified bit-rate. The problem with discarding image and video information is that after a certain point, too much information could be discarded for a human viewer to be able to understand the contents of a decoded video scene. If this were the case, a decoded video scene would be of no use even if the required bit-rate of the system were satisfied. PSNR does not provide an accurate measurement of the perceived visual distortion of a decoded video sequence as it only measures the mathematical distortion between the individual pixels of two video frames [Teo94], [Osberg97] and [Damer00]. Considering that a human is the ultimate viewer, therefore only he or she would be able to decide whether the fidelity of a decoded video stream is aesthetically acceptable or not. Therefore in order to holistically evaluate the performance of the implemented system, discussion of the obtained results will be based upon the above mathematical evaluation criteria as well as general human impression.

5.2 Reference Frame Evaluation

This section analyses the performance of the reference frame encoder and decoder. Recall that the purpose of the reference frame is to provide a background on which segmented and tracked vehicles are superimposed and displayed. In this manner, the viewed decoded video stream appears as a complete entity even though it is actually synthetically created. Thus the goal of the frame encoder is to maximise the output compression ratio whilst retaining reasonable fidelity so that on decoding, the system operator is still able to comprehend the contents of the background video scene.

Three captured video frames are used to test the performance of the implemented encoder and decoder. The video frames are illustrated in Figures 5-1(a), (b) and (c). From here on, the video frames will be referred to as VF-1 (Video Frame), VF-2 and VF-3 respectively.

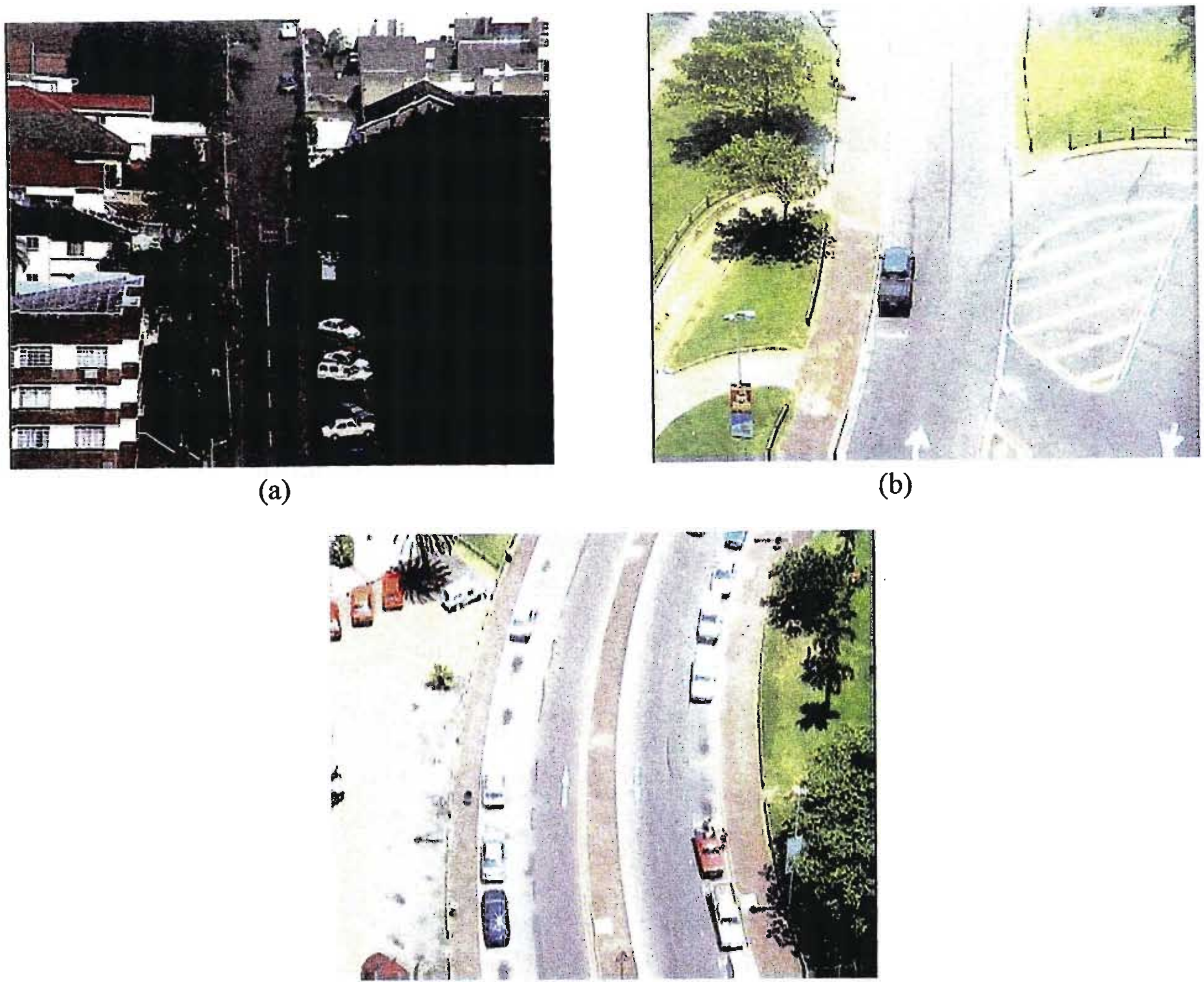

(a) (b) (c)

Figure 5-1: Video frames used for evaluation, (a) VF-1 (Video Frame), (b) VF-2 and (c) VF-3.

Although the video frames are illustrated with a reduced size, they were all captured in CIF format (352x288 pixels), sub-sampled according to the 4:2:0 ratio and saved on the PC's HDD in separate Y, U and V files.

The operation of the reference frame encoder is subject to four user-defined input variables. These variables include **Qfactor, ref_matrix_size, Thres_32** and **Thres_16**. The system was initially evaluated by varying each the four variables one at a time while the others remained fixed. Thereafter, the results were combined in order to determine a range of variable settings that would ideally satisfy the goal of the frame encoder and decoder, i.e. the maximisation of the compression ratio while still retaining reasonable video frame fidelity.

5.2.1 Variable Block-Size Sub-Division

The purpose of the implemented variable block-size algorithm is to sub-divide the Y component of an input video frame into blocks of varying size as a function of spatial detail. Recall that the U and V components of an input video frame are not sub-divided into blocks of varying sizes, instead they are sub-divided into blocks of fixed size of 16x16 pixels. Recall that if the standard deviation of a Superblock exceeds **Thres_32** then it is sub-divided in quad-tree fashion into four 16x16 blocks.

Similarly, if any of the four 16x16 block's standard deviation exceeds **Thres_16**, then they are quad-tree sub-divided into four 8x8 blocks respectively.

Figure 5-2 provides the simulated results of varying **Thres_32** independently of the three remaining user-defined variables. In order to ensure that **Thres_16** had no affect on the simulation, it was set to the unreachable value of 200. **Qfactor** was set to 100 ensuring minimal quantisation while **ref_matrix_size** was set to the maximum value of 36 thus ensuring the maximum number of remaining macroblock coefficients after DCT encoding.

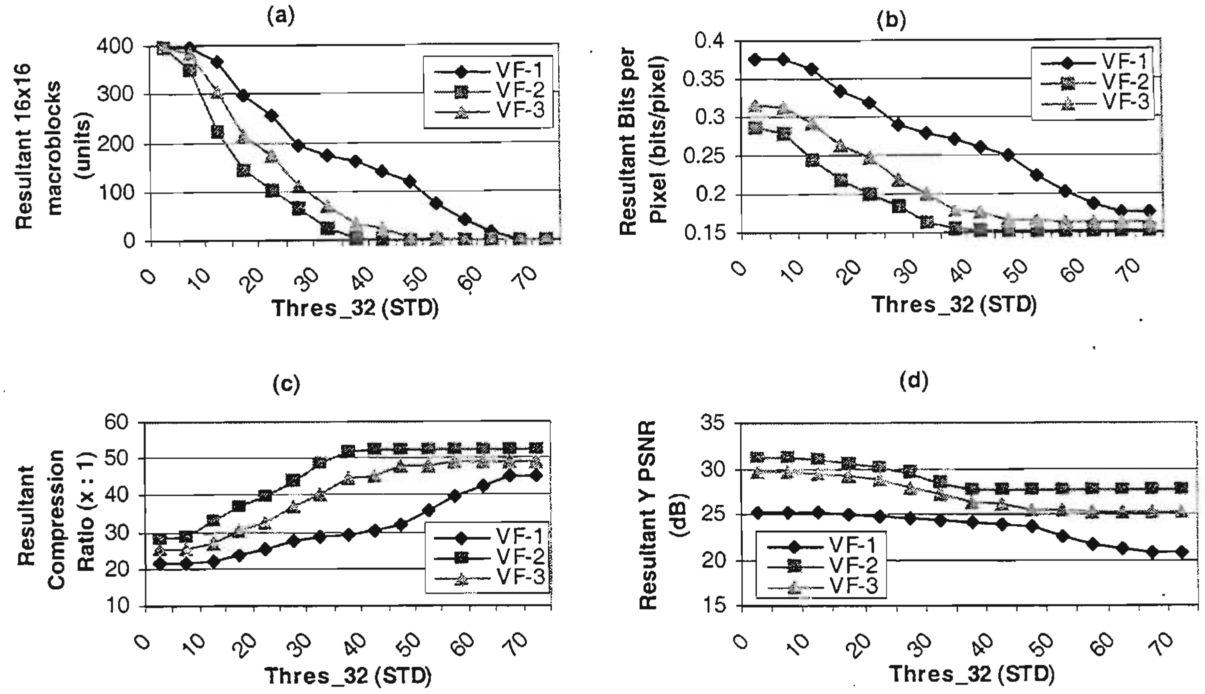


Figure 5-2: Results obtained from the variation of **Thres_32**, (a) number of 16x16 macroblocks,(b) bits per pixel, (c) compression ratio, (d) Y peak signal to noise ratio.

From Figure 5-2(a) it is noted that as the user-defined **Thres_32** threshold increases, the number of resulting 16x16 macroblocks decreases to the point where only 32x32 Superblocks remain. The resultant compression ratio, bpp and Y component PSNR values are illustrated in Figure 5-2(c), (b) and (d) respectively.

As the **Thres_32** threshold value decreases, the number of 32x32 Superblocks decreases resulting in an increase of the number of 16x16 macroblocks. As a consequence, output compression ratios decrease due to the increasing number of macroblocks to encode. Furthermore from Figure 5-2(d), it is noted that as the number of 16x16 macroblocks increases, PSNR increases as well. This results due to a smaller sized macroblocks retaining more image fidelity on decoding than larger sized macroblocks.

Another important characteristic that is noted from Figure 5-2(a) is that this particular graph gives an indication of the general spatial detail within each respective video frame. Considering that spatial detail is proportional to standard deviation, it is expected that a video frame comprising high spatial detail should be sub-divided into a greater number of smaller sized macroblocks at a faster rate compared to an image of low spatial detail. Therefore from Figure 5-2(a), it would be expected that VF-1 contains greater levels of spatial detail compared to VF-3 and VF-2. On analysing the general spatial detail within the original video frames illustrated in Figure 5-1, we can conclude that the analysis is reasonably accurate.

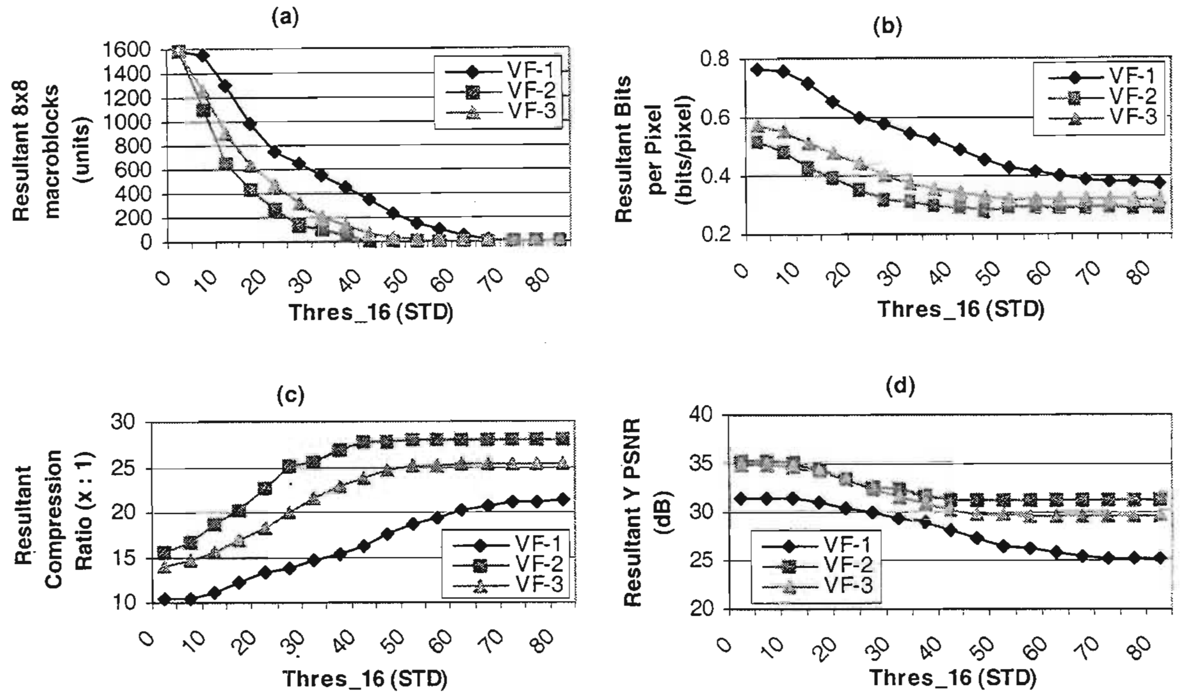
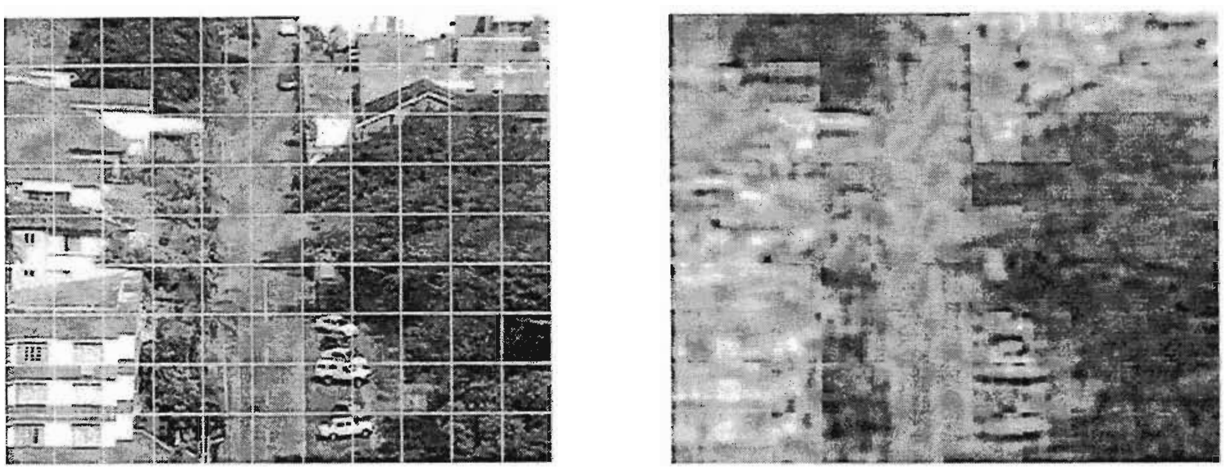
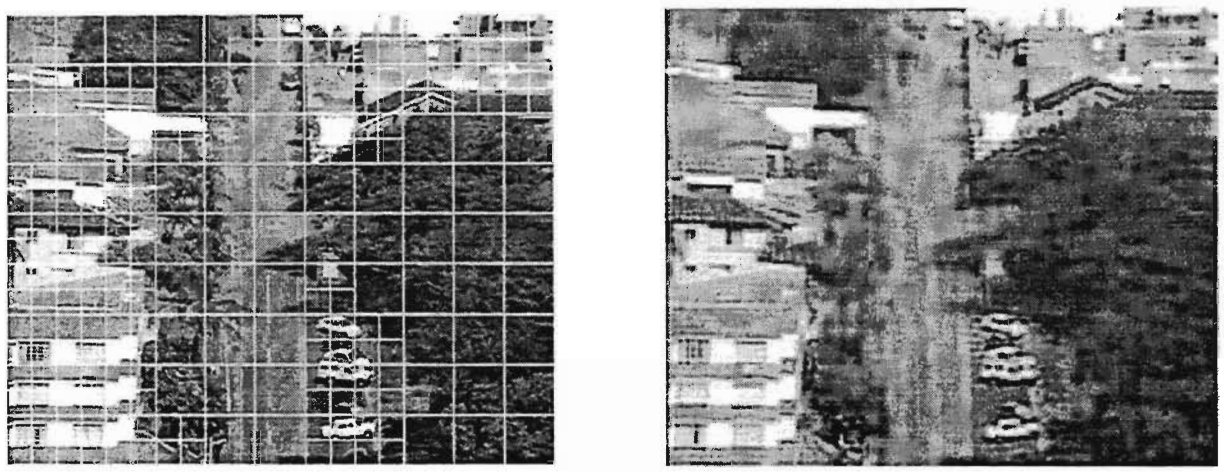


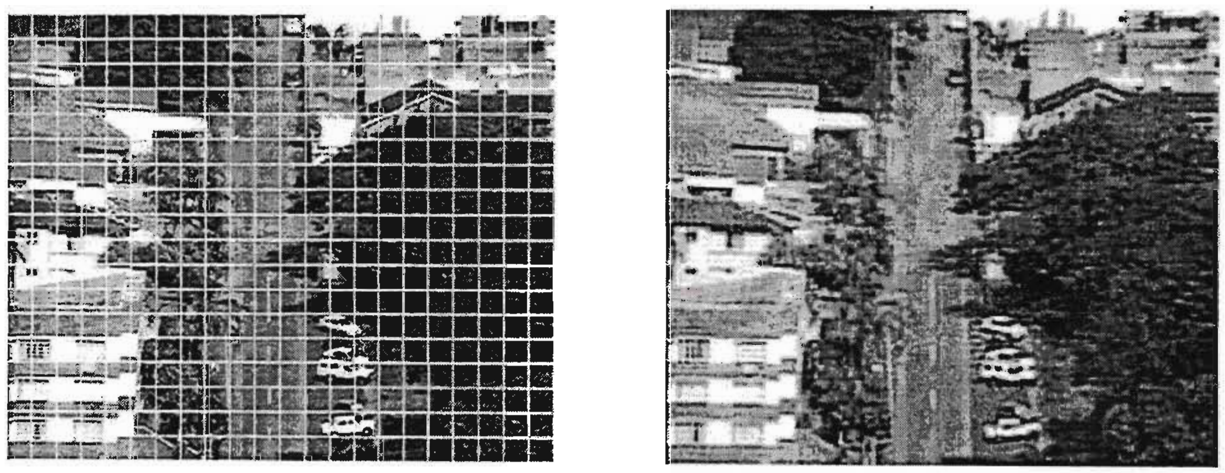
Figure 5-3: Results obtained from the variation of **Thres_16**, (a) number of 8x8 macroblocks,(b) bits per pixel, (c) compression ratio, (d) Y peak signal to noise ratio.

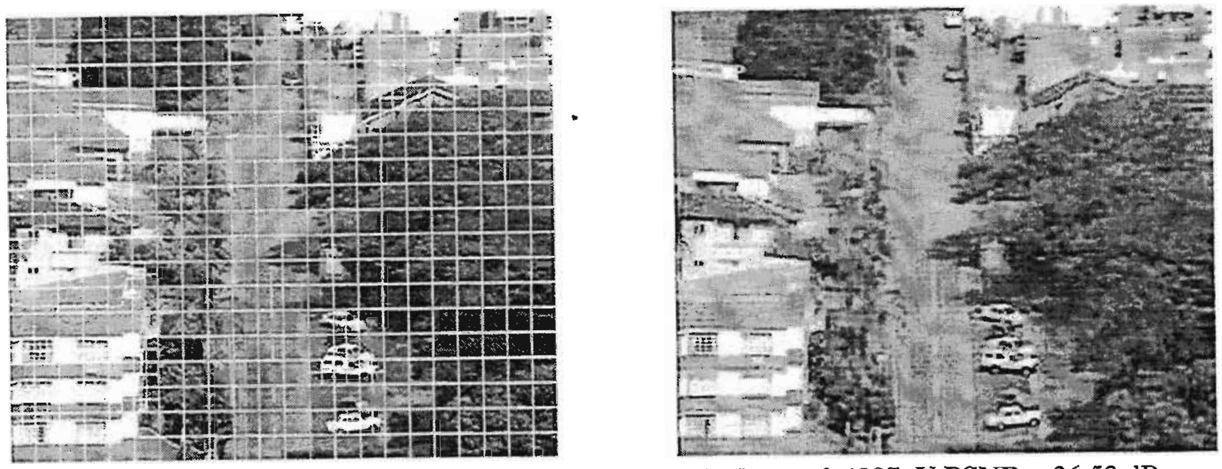
Figure 5-3 illustrates the simulated results of varying the **Thres_16** variable independently of the three remaining user-defined variables. At the outset of the simulation, **Thres_32** was set to zero therefore ensuring the maximum block size would be 16x16 pixels, **Qfactor** was set to 100 and **ref_matrix_size** was set to 36. From the results it is noted that the greater the number of encoded 8x8 macroblocks, the lower the output compression ratio of the encoded YUV video frame as would be expected. At the same time, an increased number of 8x8 macroblocks increases the resultant PSNR as 8x8 macroblocks retain more frame fidelity compared to 16x16 macroblocks.

Figure 5-4 illustrates a set of decoded video frames of the VF-1 test frame. The threshold settings are indicated below each resultant video frame along with its measured compression ratio, bpp and PSNR. For visual purposes, an extra software function was written in order to graphically display the variable block-size sub-division of the input video frame according to the user defined threshold settings. The video frames on the left side illustrate the sub-division of the input video frame into white blocks of varying size, while the video frames on the right illustrate the decoded result. Naturally the largest white blocks indicate Superblocks of size 32x32 pixels and the smallest macroblocks are of size 8x8 pixels. The results of VF-2 and VF-3 at similar threshold settings are placed in Appendix C.

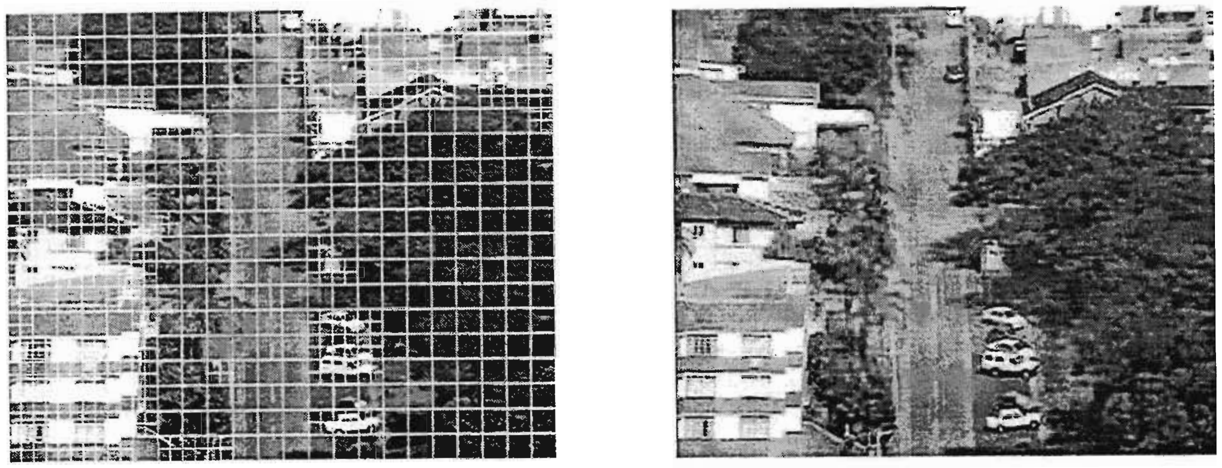

Thres_32 = 70, **Thres_16** = 200, **Compression Ratio** = 45.23, **bpp** = 0.1769, **Y PSNR** = 20.72 dB


Thres_32 = 30, **Thres_16** = 200, **Compression Ratio** = 28.76, **bpp** = 0.2782, **Y PSNR** = 24.33 dB

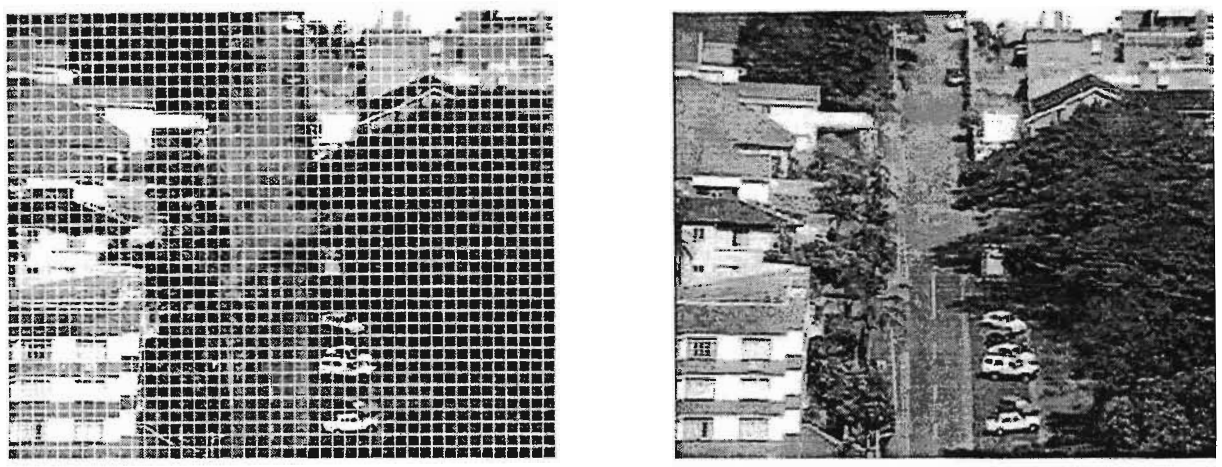

Thres_32 = 0, **Thres_16** = 200, **Compression Ratio** = 21.32, **bpp** = 0.3752, **Y PSNR** = 25.17 dB



Thres_32 = 0, **Thres_16** = 50, **Compression Ratio** = 18.62, **bpp** = 0.4297, **Y PSNR** = 26.53 dB



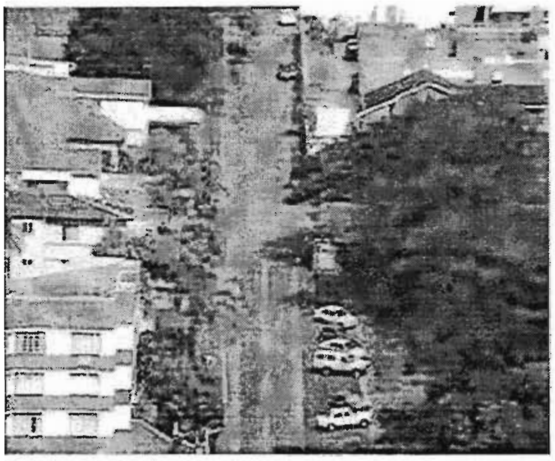
Thres_32 = 0, **Thres_16** = 30, **Compression Ratio** = 14.63, **bpp** = 0.5466, **Y PSNR** = 29.43 dB

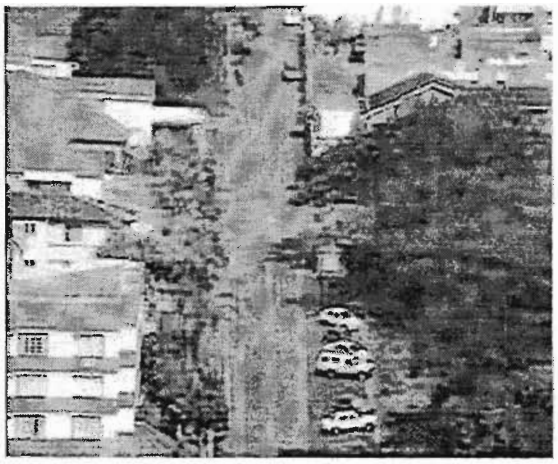


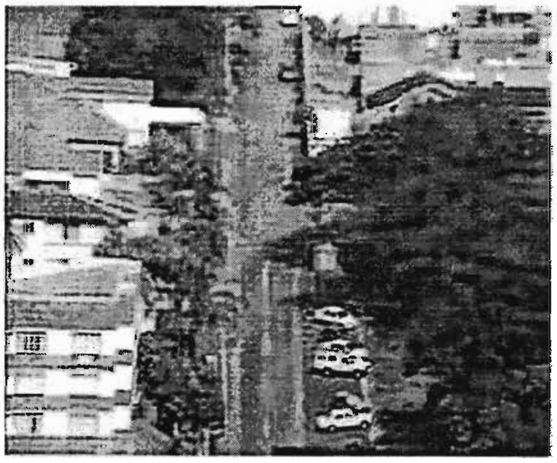
Thres_32 = 0, **Thres_16** = 0, **Compression Ratio** = 10.46, **bpp** = 0.7648, **Y PSNR** = 31.54 dB

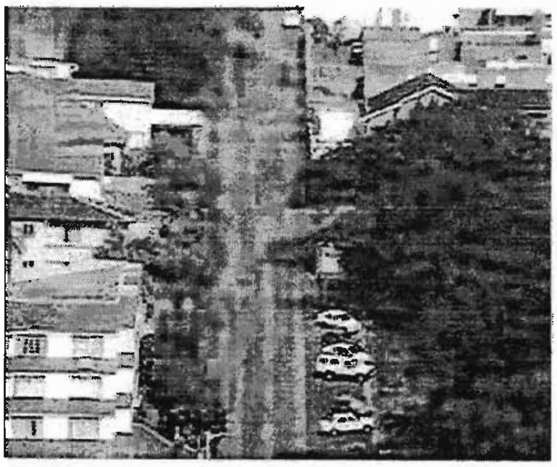
Figure 5-4: Decoded results of VF-1 while independently varying **Thres_32** and **Thres_16**.

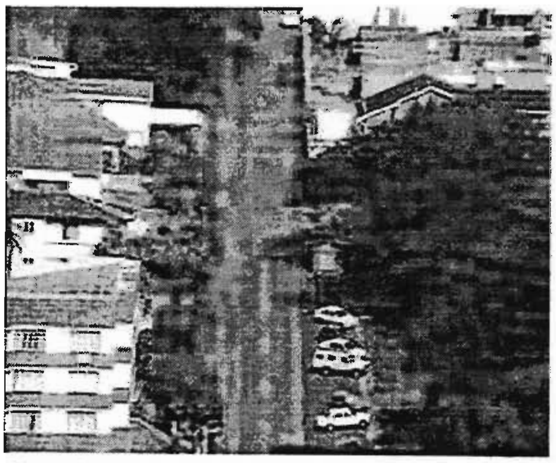
The last step in this section is the combination of the two varying thresholds. Based on the mathematical results from the simulations and human impression of the illustrated video frames in this section as well in Appendix C, it is felt that a reasonable range of the thresholds include **Thres_32** = [15, 25] and **Thres_16** = [30, 50]. Figure 5-5 illustrates some decoded examples of VF-1 that were obtained by varying the thresholds within the suggested range in relation to each other. Similar results of VF-2 and VF-3 are placed in Appendix C. The results were obtained with **ref_matrix_size** set to 36 and **Qfactor** set to 100.


Thres_32 = 15, **Thres_16** = 30, **Compression Ratio (CR)** = 15.75, **bpp** = 0.5081, **Y PSNR** = 29.00 dB


Thres_32 = 15, **Thres_16** = 40, **CR** = 17.76, **bpp** = 0.4505, **Y PSNR** = 27.81 dB


Thres_32 = 15, **Thres_16** = 50, **CR** = 20.48, **bpp** = 0.3905, **Y PSNR** = 26.31 dB


Thres_32 = 25, **Thres_16** = 30, **CR** = 17.55, **bpp** = 0.4559, **Y PSNR** = 27.71 dB


Thres_32 = 25, **Thres_16** = 40, **CR** = 19.70, **bpp** = 0.4060, **Y PSNR** = 26.88 dB

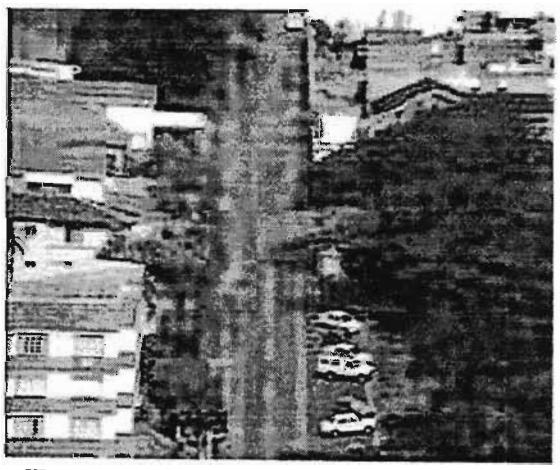

Thres_32 = 25, **Thres_16** = 50, **CR** = 23.16, **bpp** = 0.4345, **Y PSNR** = 25.63 dB

Figure 5-5: Decoded results of VF-1 while simultaneously varying **Thres_32** and **Thres_16.**

5.2.2 *Effects of Varying ref_matrix_size*

Recall that the user-defined **ref_matrix_size** variable determines the number of macroblock coefficients that remain following application of the DCT. It can take on any value from the following set: {36, 28, 21, 15, 10, 6, 3, 1} and it affects all three YUV components of a video frame. Figures 5-6 and 5-7 provide the simulated results of VF-1 as a function of **ref_matrix_size**. **Thres_32** and **Thres_16** were set to predefined values while **Qfactor** was set to 100.

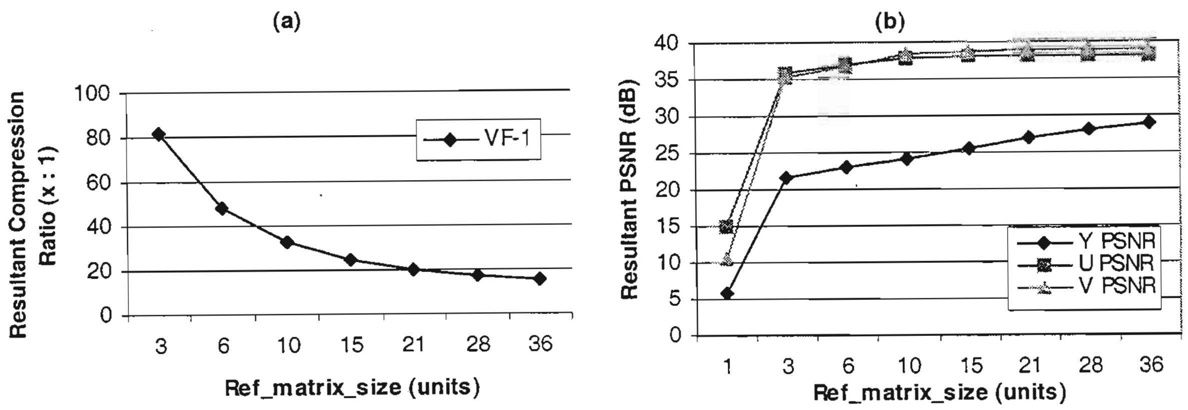


Figure 5-6: VF-1 results obtained from the variation of **ref_matrix_size** with **Thres_32** = 15 and **Thres_16** = 30, (a) compression ratio and (b) peak signal to noise ratio.

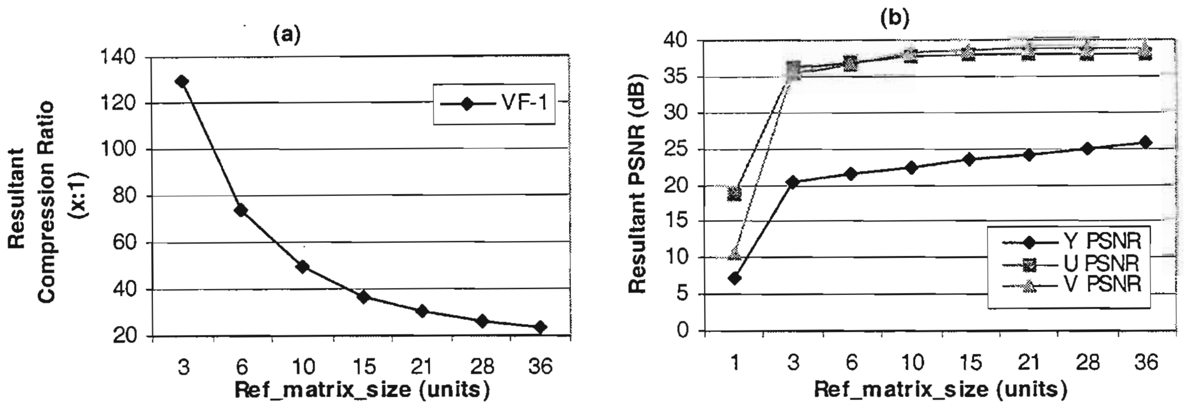
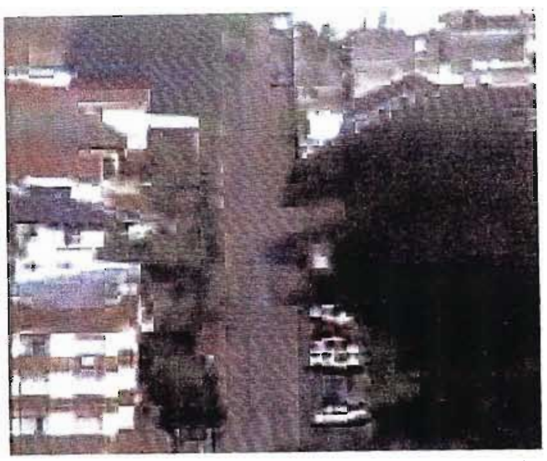
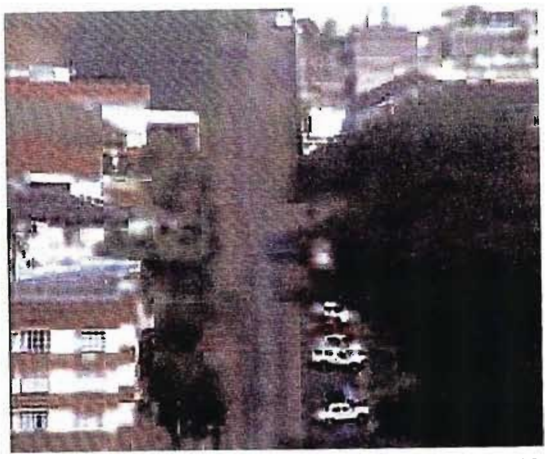
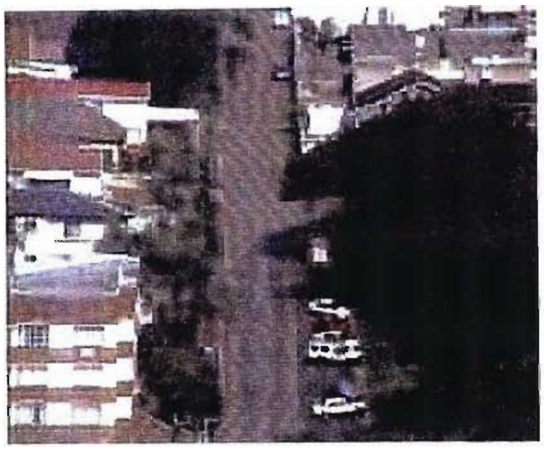


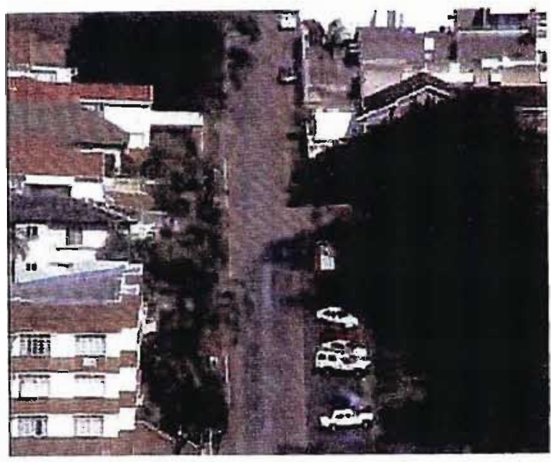
Figure 5-7: VF-1 results obtained from the variation of **ref_matrix_size** with **Thres_32** = 25 and **Thres_16** = 50, (a) compression ratio, and (b) peak signal to noise ratio.

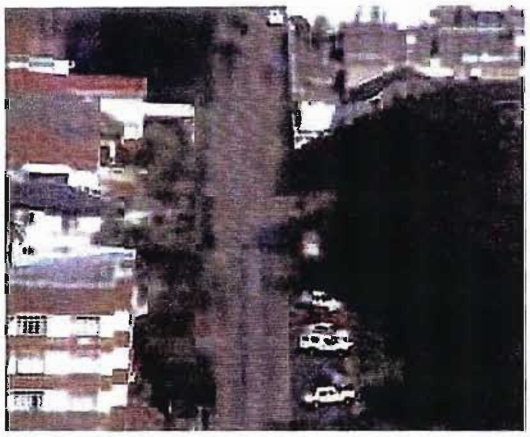
In both Figures 5-6(a) and 5-7(a), the resultant compression ratios with **ref_matrix_size** set to one have purposefully been omitted from the respective graphs, as they are far too large in value. The values respectively are 228.32 and 362.70. The results prove the expectations in that as **ref_matrix_size** increases, the quality of the decoded frame, given by PSNR, increases at the expense of compression ratio. Figure 5-8 illustrates a selected number of the decoded frames as a function of **ref_matrix_size**.


Thres_32 = 15, Thres_16 = 30, ref_matrix_size = 6


Thres_32 = 15, Thres_16 = 30, ref_matrix_size = 10


Thres_32 = 15, Thres_16 = 30, ref_matrix_size = 15


Thres_32 = 15, Thres_16 = 30, ref_matrix_size = 28


Thres_32 = 25, Thres_16 = 50, ref_matrix_size = 15

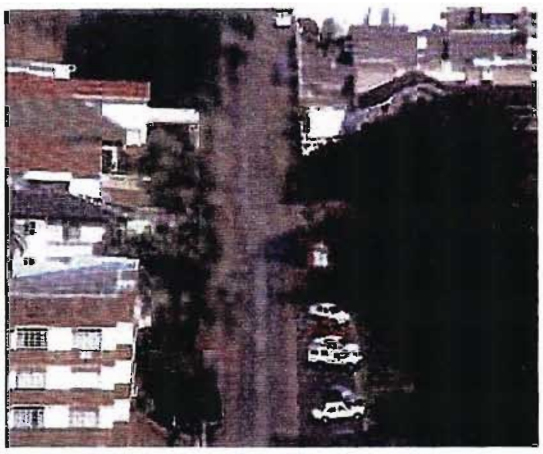

Thres_32 = 25, Thres_16 = 50, ref_matrix_size = 36

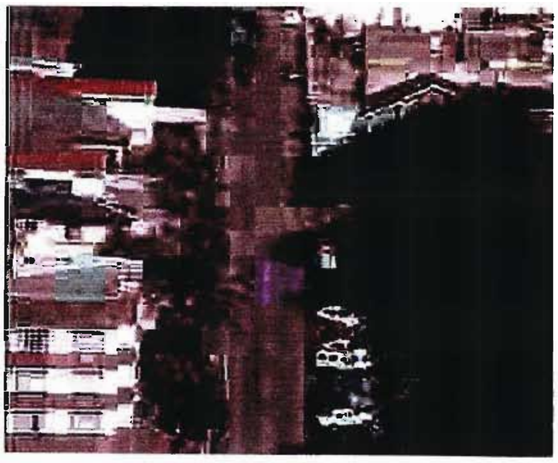
Figure 5-8: Decoded results of VF-1 as a function of **ref_matrix_size.**

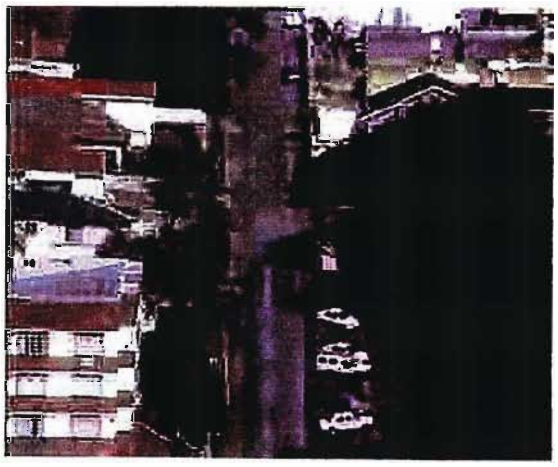
From the results presented here as well as those in Appendix C, a reasonable range of **ref_matrix_size** includes the values from 15 up to 28. These values were chosen as they generally resulted in the fidelity of the decoded frames ranging from slightly blurry to reasonably clear hence optimising compression ratio.

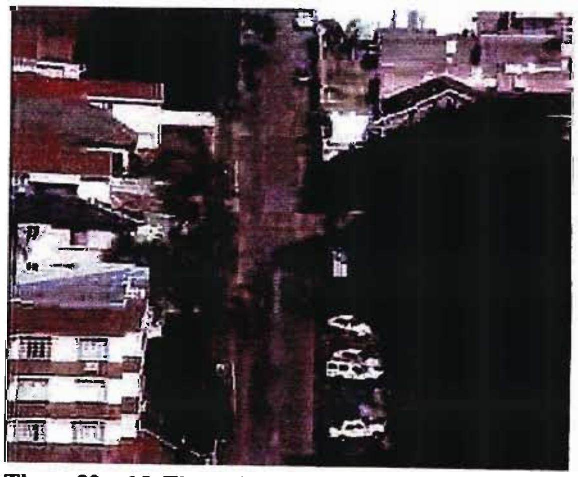
5.2.3 Effects of Varying Qfactor

Recall from Chapter 4 that **Qfactor** results in a scaled factor being multiplied to each coefficient within the YUV quantisation tables. Each transformed macroblock coefficient is then divided by its corresponding scaled quantisation coefficient.

The resultant decoded video frames of VF-1 subject to the variation of **Qfactor** are illustrated in Figure 5-9. From the results it is noted that even with **Thres_32** and **Thres_16** set to the minimum values within their suggested ranges and **ref_matrix_size** set to a reasonably high value of 21, there is a significant amount of colour distortion within the decoded video frames with **Qfactor** $\leq$ 97. Therefore from the results, a minimum value of **Qfactor** to be used in conjunction with the suggested range of thresholds and **ref_matrix_size** should be at least 98. This minimum value would ensure that the resultant colour distortion within a decoded video frame remains aesthetically acceptable.


Thres_32 = 15, **Thres_16** = 30, **ref_matrix_size** = 21, **Qfactor** = 95, **CR** = 46.42, **Y PSNR** = 23.75 dB, **U PSNR** = 31.22 dB , **V PSNR** = 29.76 dB


Thres_32 =15, **Thres_16** =30, **ref_matrix_size** =21, **Qfactor** = 97, **CR** = 37.65, **Y PSNR** = 25.11 dB, **U PSNR** = 33.05 dB , **V PSNR** = 33.91 dB


Thres_32 = 15, **Thres_16** = 30, **ref_matrix_size** = 21, **Qfactor** = 98, **CR** = 32.06, **Y PSNR** = 25.89 dB, **U PSNR** = 35.23 dB , **V PSNR** = 35.52 dB

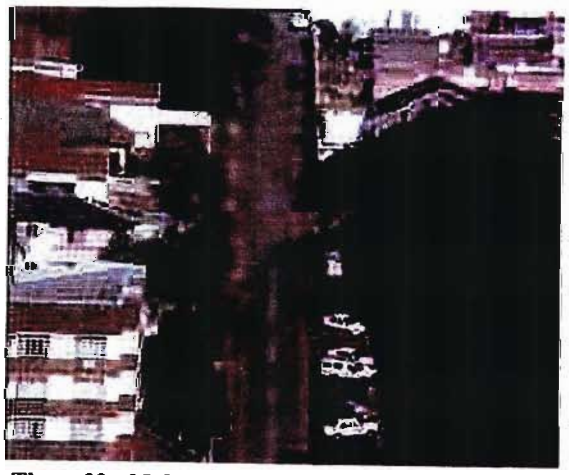

Thres_32 =25, **Thres_16** =50, **ref_matrix_size** =36, **Qfactor** = 98, **CR** = 37.59, **Y PSNR** = 24.77 dB, **U PSNR** = 35.23 dB , **V PSNR** = 35.52 dB

Figure 5-9: Decoded results of VF-1 as a function of **Qfactor**.

5.2.4 Consolidated Results

The four user-defined variables are now varied simultaneously in relation to each other in order to determine their combined affect with regard to fidelity of a decoded video frame. The results illustrated in Figure 5-10 compare the resultant Y PSNR ratios of the three test video scenes as a function of encoding efficiency (bpp).

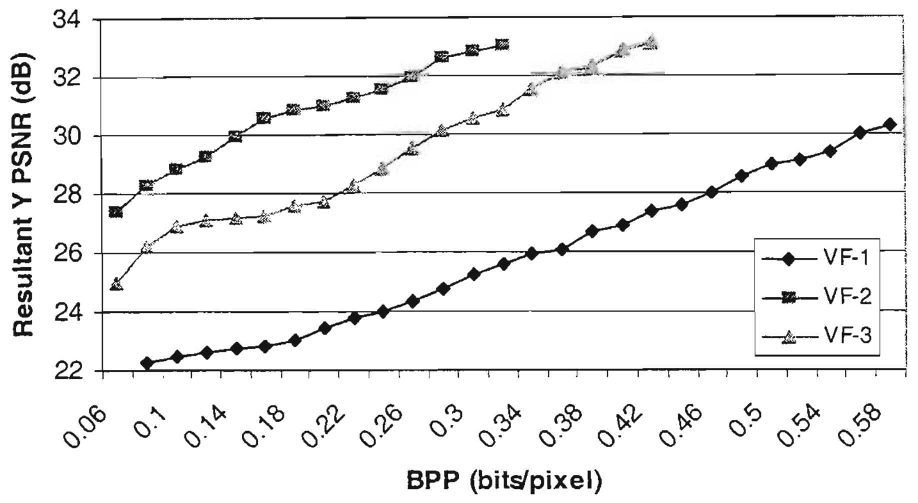


Figure 5-10: Resultant Y PSNR vs. bpp.

Besides the expected result that as bpp decreases so the resultant Y PSNR decreases as well, what is distinct is the difference between the results obtained for VF-1 and VF-2. This difference is attributed to the vastly different levels of spatial detail within the test frames. Naturally within generally low detail images such as VF-2, less spatial information is discarded when the frame is highly compressed compared to high spatial images such as VF-1. This is an unfortunate characteristic of the algorithm, which is inherent of the JPEG standard as well, in that encoding efficiency is very much dependent on the content of the compressed image. The only compensation for this downfall is that the system operator has the ability to control the variables that directly affect the compression levels. However this functionality does not necessarily guarantee an optimally encoded image all the time.

5.2.5 Time Measurements

Figure 5-11(a) illustrates the measured time required to encode each of the three test video frames as a function of bpp. The results in milliseconds do not take into account the time required to capture and digitise an input video frame from the streaming input. It is noted that the required encoding time is almost linear with respect to an increasing bpp. This is expected as an increasing bpp generally corresponds to an increasing number of smaller sized macroblocks, thus the greater the number of macroblocks to encode, the greater the processing time required.

However on inspecting the results closely, it is noted that VF-1 is generally encoded quicker than both VF-2 and VF-3. Considering VF-1 contains greater spatial detail, it would be expected that it should take longer to encode than the other test video frames that contain less spatial detail. In order to explain this result, it was necessary to analyse the timing innards of the encoding algorithm. The most time consuming stage during encoding is the DCT. A simulation was executed in order to determine the speed of the implemented 1D 8, 16 and 32 DCT's in relation to each other. The simulation involved initialising each of the different sized arrays with random values and recording the time required to transform the arrays. After 800 executions, an average transformation time was calculated per array. The results indicated that a 1D 16 element array was transformed 2.84 times slower than a 1D 8 element array and that a 1D 32 element array was transformed 11.25 times slower than a 1D 8 element array.

With these results in mind, in order to analyse the affects the DCT had on the total encoding time per test frame, four bpp values and their resulting encoding times were randomly selected from the results used to plot Figure 5-11. These results are presented in Table 5-1 and they include the **ref_matrix_size** variable of each encoded frame, the resulting number of variable sized blocks that the Y component was sub-divided into and the representative DCT encoding times.

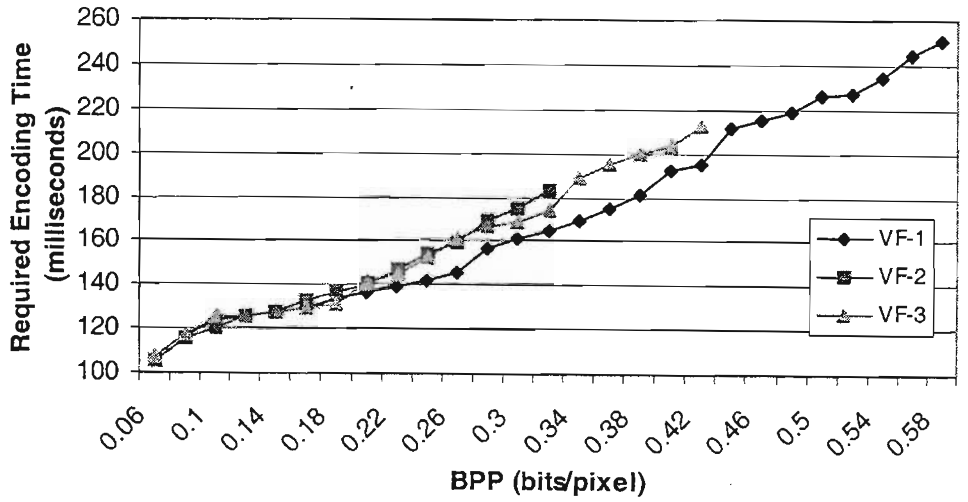


Figure 5-11: Required test video frame encoding time.

Table 5-1: Comparison of encoding results of the test video frames.

			Number of blocks:			Representative DCT Times:
	bpp (bits/pixel)	**ref_matrix_size**	**32x32**	**16x16**	**8x8**	
VF-1	0.18	10	35	196	240	28,187
VF-2		36	43	191	132	34,480
VF-3		10	56	70	128	28,192
VF-1	0.22	10	35	143	452	27,721
VF-2		21	63	81	252	35,521
VF-3		21	56	140	128	34,479
VF-1	0.24	10	7	255	452	22,743
VF-2		36	63	118	104	38,056
VF-3		36	56	169	12	36,911
VF-1	0.32	15	25	160	544	27,020
VF-2		36	43	158	264	34,343
VF-3		36	23	272	128	30,937

The purpose of including the **ref_matrix_size** variable within the table is to illustrate the number of 1D DCT's that were executed per block per encoded frame. As an example consider the tabulated results of VF-1 with bpp = 0.18. In this case, the Y component was sub-divided into 35 32x32, 196 16x16 and

240 8x8 blocks. In order to calculate the affect that the number of executed DCT's had on the overall encoding time of this particular frame, we calculate the number of 1D DCT's that were executed per block. Recall that the implemented 2D DCT algorithm begins by applying a 1D DCT to every row of a block to be transformed, and then only applies a certain number of 1D DCT's to the columns of the block as determined by the user-defined **ref_matrix_size**. A **ref_matrix_size** value of ten corresponds to four 1D DCT's applied to the columns of a block, a value of 15 corresponds to five 1D DCT's applied and so on with a maximum **ref_matrix_size** of 36 corresponding to eight 1D DCT's being applied.

Going back to the example with **ref_matrix_size** set to 10, a total of (32 + 4)*35 = 1,260 32 element 1D DCT's, (16 + 4)*196 = 3,920 16 element 1D DCT's and (8 + 4)*240 = 2,880 8 element 1D DCT's were executed during the transformation of the Y component. We then calculate a value that represents the total required DCT encoding time by summing the times required to transform the 8, 16 and 32 element 1D DCT's. Considering that the times taken to transform the 32 and 16 element 1D DCT's were calculated as a factor of the time taken to transform an 8 element 1D DCT, the total representative time of the DCT encoding stage is calculated from (1,260)(11.25) + (3,920)(2.84) + (2,880) = 28,187. The representative times of the remaining encoded frames are calculated in a similar manner and presented in the last column in Table 5-1. By comparing the representative values of the different test frames per bpp, it gives an indication of the amount of time that was spent during the DCT stage. On comparing the results of the different test frames for bpp = 0.18, the representative times of VF-1 and VF-2 are similar and also lower than that of VF-2. Therefore based on these results it would be expected that the total encoding time of VF-1 and VF-3 should be lower than VF-2. On analysing Figure 5-11 it is noted that this is in fact the case albeit that the difference is not very pronounced. On comparing the remaining representative times in Table 5-1 to the results in Figure 5-11, it is noted that proportionately they are reasonably accurate. Slight discrepancies are attributed to the varying times that resulted during the other encoding stages.

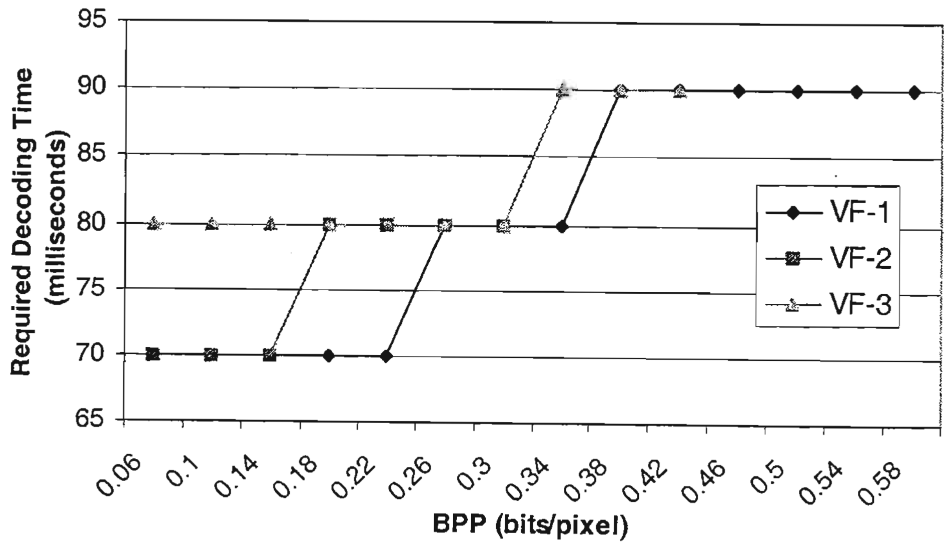


Figure 5-12: Required test video frame decoding time.

The required decoding times are illustrated in Figure 5-12. Recall that the decoder is executed on the host PC, which in this case is a Pentium 3 with the CPU clocked at 667 MHz. Unfortunately the results were not able to be measured in as fine resolution as the encoding times. The reason for this was that the provided *clock* functions within MSVC++ are not specifically geared towards the measurement of millisecond intervals. Thus from the results, the only significant fact that can be obtained from the results is that in all three cases, the test video frames are decoded quicker than they are encoded.

5.2.6 *Performance Compared to JPEG*

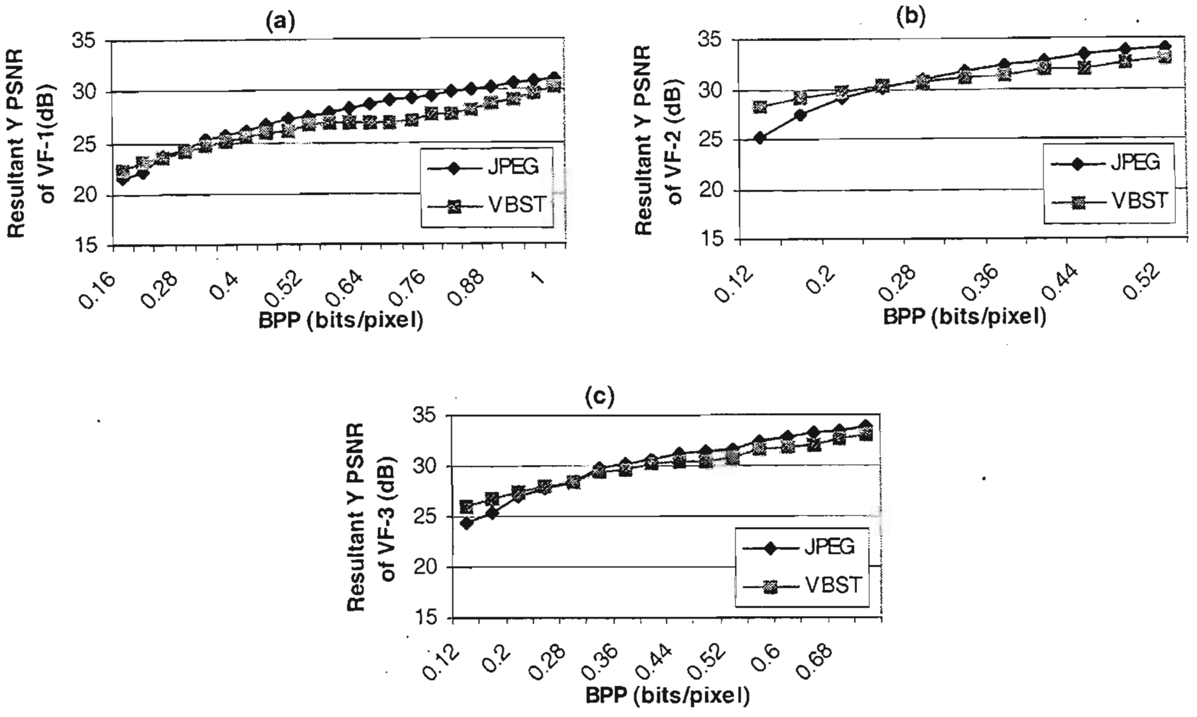


Figure 5-13: Comparison to JPEG applied to test video frames (a) VF-1, (b) VF-2 and (c) VF-3.

Figure 5-13 illustrates the results obtained from the comparison of the implemented algorithm to the JPEG still image compression standard. Note that 'VBST' in the legend within the graphs refers to the implemented Variable Block-Size Transform algorithm. The graphs illustrate the resultant Y PSNR of the three difference test video frames as a function of bpp. With regard to the JPEG results, the bpp measurements were taken by use of the image processing program Paint Shop Pro (PSP) ver 4.12. PSP allows a user to set the number of Dots Per Inch (DPI) as well as the compression level of a JPEG compressed image. The higher the compression level, the higher the output compression ratio hence the lower the resulting bpp. In this case, DPI was set to 300 and the compression level was varied within the range [1, 99].

In order to compare JPEG compressed images to those of the implemented system, the first step was to approximately match the bpp values of the two differently compressed frames. Once matched, the JPEG video frame was then directly converted to RAW format and saved to HDD. A software program was written to calculate the PSNR ratio between the converted RAW-JPEG video frame and its original Y component. The calculated PSNR value at that particular bpp value was then tabulated. This measurement technique was then applied over again according to a predetermined range of bpp values. The PSNR vs. bpp results were then graphed.

The U and V PSNR values were not calculated as recall that the original test video frames are sub-sampled according to the 4:2:0 ratio. Considering the JPEG compression tool offered by PSP can only compress colour images that are in 4:4:4 format, the U and V components of the original test video frames would have to be upsampled. The technique of up-sampling technique would not give a fair indication of the degradation that resulted during encoding, therefore it was decided not to compare the U and V PSNR values of the different systems.

On analysing the above graphs, the performance of the implemented encoder is slightly lower than that of the JPEG still image compression standard at bpp values greater than 0.28 bits/pixels. For any given bpp value within all three of the graphs depicted in Figure 5-13, the resulting PSNR ratio of the implemented encoder is at worst approximately 3 dB's less than that of the JPEG frames. The performance of the implemented encoder only ever exceeds that of the JPEG encoder at low bpp values, i.e. bpp values $\leq$ 0.28 bits/pixel.

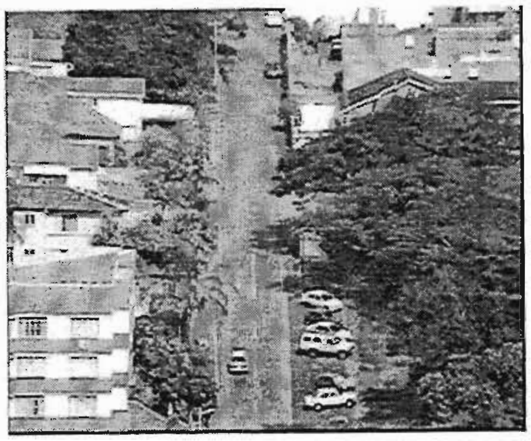
The reason for this observation is that at the higher bpp values, there are more bits to allocate per encoded block. Considering the JPEG images, because each block is limited to 8x8 pixels in size, the allocation of a larger number of bits per block would result in greater retention of spatial detail and ultimately in increased image quality. On the other hand, the implemented system divides an image into variable sized blocks and combined with the utilised coefficient limiting algorithm, there is always a limited number of bits that can be allocated per block, irrespective of its size. Because of this, it would be expected that the larger sized blocks would have far more distortion than the smaller sized blocks and this would ultimately impact on the image's PSNR in a negative manner.

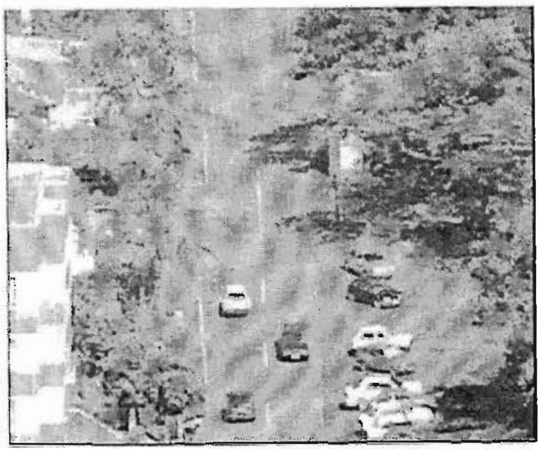
However at bpp values $\leq$ 0.28 bits/pixel, there would be fewer bits to allocate per encoded block overall. Furthermore, as the implemented system would have fewer blocks to encode than JPEG, more bits are allocated per block hence the retained spatial detail is greater than that of the JPEG standard. Therefore it would be expected that the PSNR values of the implemented system would be greater than their JPEG counterparts.

5.3 Object Segmentation and Tracking Evaluation

This section analyses the performance of the implemented object segmentation and tracking algorithms. Recall the goal of the implemented object segmentation and tracking algorithms is to maximise their ability to accurately segment and track objects within a test video scene and in doing so optimise output bit-rate.

Three test video sequences were used to evaluate the performance of the segmentation and tracking algorithms. These test video sequences were specifically chosen as they depict different sized vehicles moving at different speeds. Sample frames of the test video sequences are illustrated in Figure 5-14. A selected number of target video frames of the three test video sequences are placed in Appendix D.


(a)


(b)

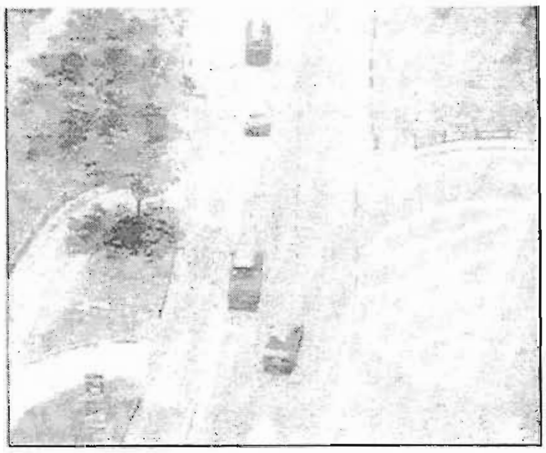

(c)

Figure 5-14: Samples of the video sequences used for performance evaluation, (a) VS-1, (b) VS-2 and (c) VS-3.

Recall that with each processing iteration, the segmentation and tracking algorithms operate on the difference of a sequence of five input video frames with the middle frame (Y_{n+2}) being buffered for object extraction purposes. This buffered frame was termed the target video frame. From here onwards, the test video sequences are be referred to as VS-1 (Video Sequence), VS-2 and VS-3 respectively.

VS-1 is sixteen seconds in length and thus corresponds to a total number of 400 captured video frames (captured at 25 frames/second). VS-2 is fourteen seconds in length thus corresponding to 350 captured video frames while VS-3 is approximately eight seconds in length thus corresponding to approximately 200 captured video frames.

Table 5-2: Segmentation and tracking user-defined variables.

ClusMap_Thres	Cluster map threshold used to filter noise in preliminary clustering step
ClusBlock_Thres	Cluster block threshold filters noise cluster blocks from object cluster blocks
Mean_Diff	Threshold used for the matching of halted objects within the video scene
ref_matrix_size	Limits the number of remaining frequency co-efficients after applying the DCT
track_boundary_top	Top co-ordinate of used defined area wherein objects are segmented and tracked
track_boundary_bottom	Bottom co-ordinate of used defined area wherein objects are segmented and tracked
track_boundary_left	Left co-ordinate of used defined area wherein objects are segmented and tracked
track_boundary_right	Right co-ordinate of used defined area wherein objects are segmented and tracked

The operation of the segmentation and tracking algorithms are subject to the eight user-defined variables depicted in Table 5-2. Within each of the following sections, the effects of varying the user-defined variables are analysed.

5.3.1 Segmentation Performance as a Function of ClusMap_Thres

Recall that the purpose of the **ClusMap_Thres** variable is to filter random noise in the preliminary stages of cluster segmentation. Having constructed a Y component difference map, the first step of cluster segmentation is the construction of a **Yclustermap**. A clustermap is constructed by sub-dividing the Y difference map into 8x8 macroblocks and mapping the mean values of each macroblock to the clustermap. Non-zero valued cluster elements signify possible areas of motion within the video scene, thus the function of the cluster segmentation algorithm is to group localised similar-valued cluster elements into cluster blocks. The co-ordinates of each cluster block are then mapped back to the target video frame. The net result is the segmentation of an object in motion within the video scene.

Figure 5-15 illustrates the segmentation results obtained from two randomly chosen target video frames from the test sequence VS-1. In this case, with respect to the target video frames in Appendix D, the selected frames were nine, illustrated in Figure 5-15(a), and 32 illustrated in Figure 5-15(b). For demonstration purposes, a software function was written to bound all segmented objects within the video scene in white rectangular blocks according to their segmented co-ordinates.

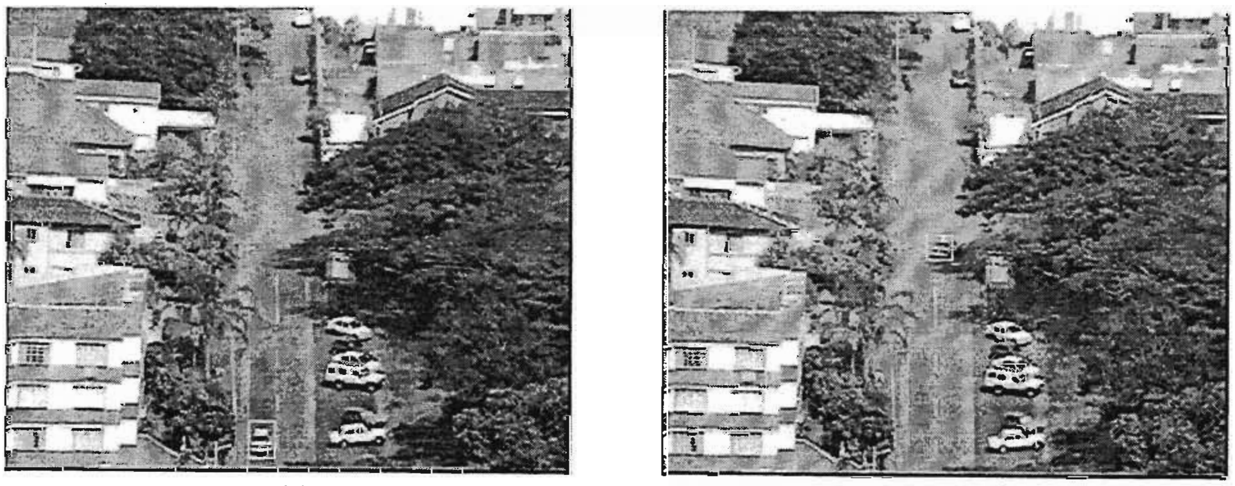
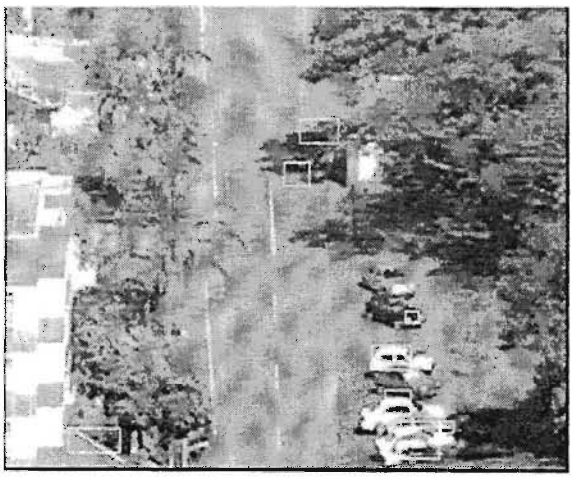

(a) (b)

Figure 5-15: Segmentation results of VS-1.

During testing, **ClusBlock_Thres** was set to zero while the **ClusMap_Thres** threshold value was initially set to zero and then increased. The test results indicated that the object was correctly segmented with a **ClusMap_Thres** value in the range [0, 39]. From the value 40 onwards, no objects within the video scene were segmented. Figure 5-15(b) illustrates the segmentation results of target video frame 32. Once again, **ClusBlock_Thres** was fixed at zero while **ClusMap_Thres** was increased from zero. The object was segmented in this case with a **ClusMap_Thres** value in the range [0, 21]. It is also noted that in this case, only the back half of the white vehicle was segmented correctly while the front portion was not included. It is possible that this results due to the front portion of the vehicle falling within the shadow of a tree. The reoccurrence of this potential shortcoming will be analysed in subsequent results.

Figure 5-16 illustrates the results obtained from applying the implemented segmentation algorithm to target video frame three of the test video sequence VS-2. Once again, **ClusBlock_Thres** was initially set to zero and then increased. Figure 5-16(a) illustrates the segmentation results when **ClusMap_Thres** was set to ten. From the illustration, it is noted that nine objects in total are segmented, with most of them being concentrated in the lower right corner of the video scene surrounding the parked vehicles. The erroneous segmented objects resulted due to the significant levels of noise within the video sequence. It is also noted the black vehicle in the shadow of the overhanging trees has been erroneously segmented as two independent objects. Figure 5-16(b) illustrates the segmentation results obtained with **ClusMap_Thres** set to twenty. In this case only three objects have been segmented, compared to the previous nine. For **ClusMap_Thres** values greater than 21, the black vehicle was not segmented at all.


(a)

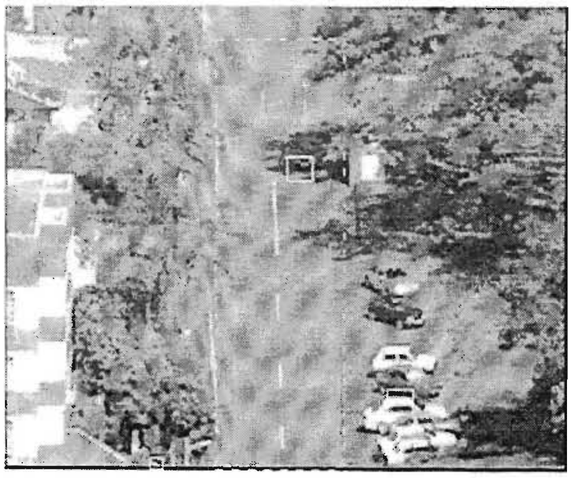

(b)

Figure 5-16: Segmentation results of VS-2.

Recall that a user is able to define a specific object segmentation and tracking area within a video scene in order to reduce erroneous object segmentation. The white dashed rectangular block in Figure 5-16(b) indicates the defined bounded tracking area used in this video scene. With the inclusion of the bounding area, the two erroneously segmented objects fall outside of the defined area and are ultimately filtered as a result. The incomplete segmentation of the black vehicle in the shadow of the overhanging trees is once again noted.

Figure 5-17 illustrates the segmentation results obtained from applying the segmentation algorithm to target video frame eleven of VS-2. With **ClusBlock_Thres** set to zero and for **ClusMap_Thres** values in the range [0, 33], the three vehicles within the defined bounding region were segmented correctly. Figure 5-17(b) illustrates the segmentation results when **ClusMap_Thres** set to 34. Note from the illustration that the lower black vehicle is not segmented at all and the right black vehicle is only partly segmented. This results due to **ClusMap_Thres** being too high in value.

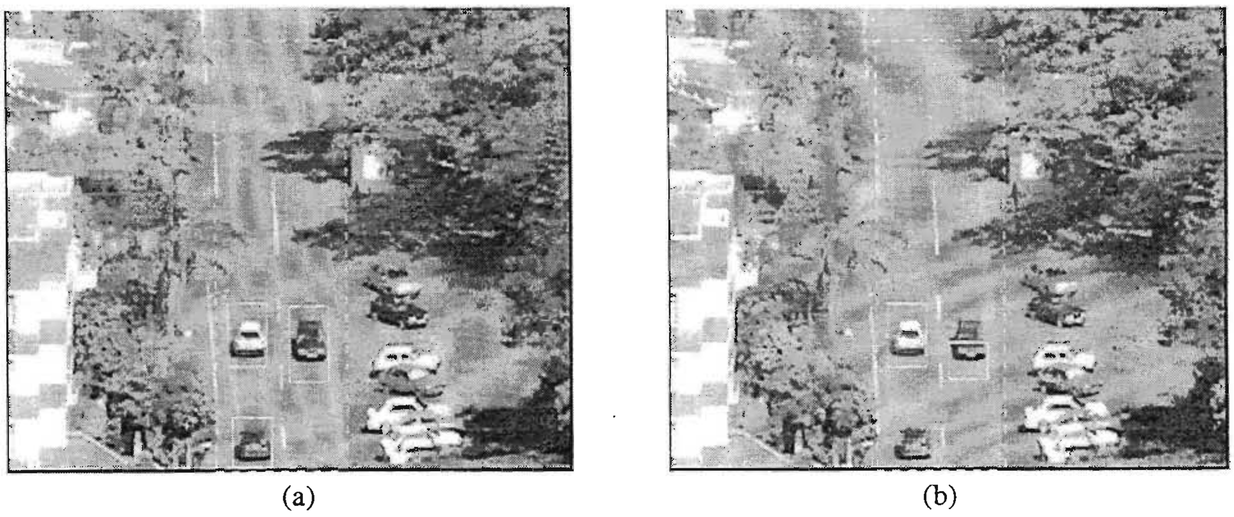


Figure 5-17: Segmentation results of VS-2 within the defined bounding area.

Figure 5-18 illustrates the results obtained from applying the segmentation algorithm to target video frames four and thirteen of VS-3. The user-defined segmentation and tracking area within this case included only the stretch of vertical road along which the object of interest was travelling. Figure 5-18(a) illustrates the segmentation results of target video frame four. The vehicle in the lower left quadrant of the frame was segmented correctly for **ClusMap_Thres** values in the range [0, 15]. Values greater than fifteen resulted in the object being partially segmented while values greater than 34 resulted in the object not being segmented at all. Figure 5-18(b) illustrates the segmentation results of target video frame thirteen. For **ClusMap_Thres** values in the range [0, 17], two objects within the scene were segmented. These include the correct object in motion towards the top of the video scene and the other at the back window of the parked vehicle on the side of the road. **ClusMap_Thres** values greater than seventeen resulted in the erroneous object being filtered while values greater than 36 resulted in no objects being segmented at all.

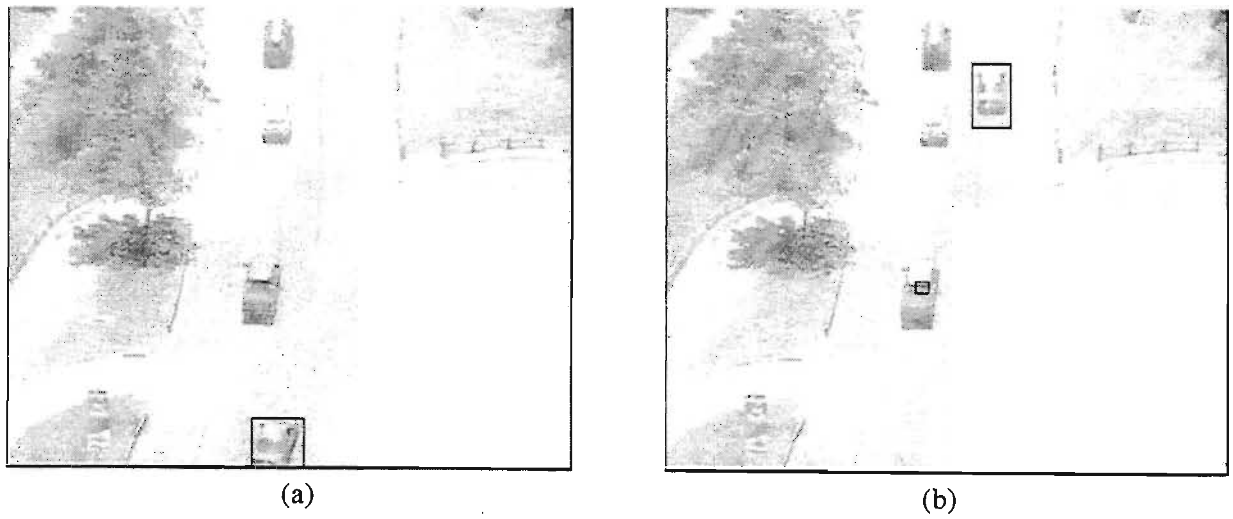


Figure 5-18: Segmentation results of VS-3.

The results suggest that the implemented algorithm is reasonably accurate in terms of segmenting objects in motion within the test video scenes. Based on the results obtained, a suitable **ClusMap_Thres** threshold that ensures that all objects are segmented correctly is fifteen. However the results thus far also suggest that in terms of noise filtering, the threshold filtering technique is not as effective as the user-defined bounding area when it comes to ambient noise within the video scene.

5.3.2 Segmentation Performance as a Function of ClusBlock_Thres

The purpose of the **ClusBlock_Thres** variable is to filter noise within the Y clustermap. Based on the assumption that noise cluster blocks comprise cluster elements that are random in magnitude and that their size tends to be unpredictable when compared to valid objects, filtering is achieved by thresholding the calculated mean values.

Figure 5-19 illustrates the segmentation results of VS-1, VS-2 and VS-3 as a function of **ClusBlock_Thres**. In all three cases, the size of the user-defined bounding segmentation and tracking area was set to the complete video scene. This was done in order to observe the noise filtering effectiveness of **ClusBlock_Thres** in conjunction with **ClusMap_Thres**. During testing, **ClusMap_Thres** was set to fifteen as suggested in the previous section.

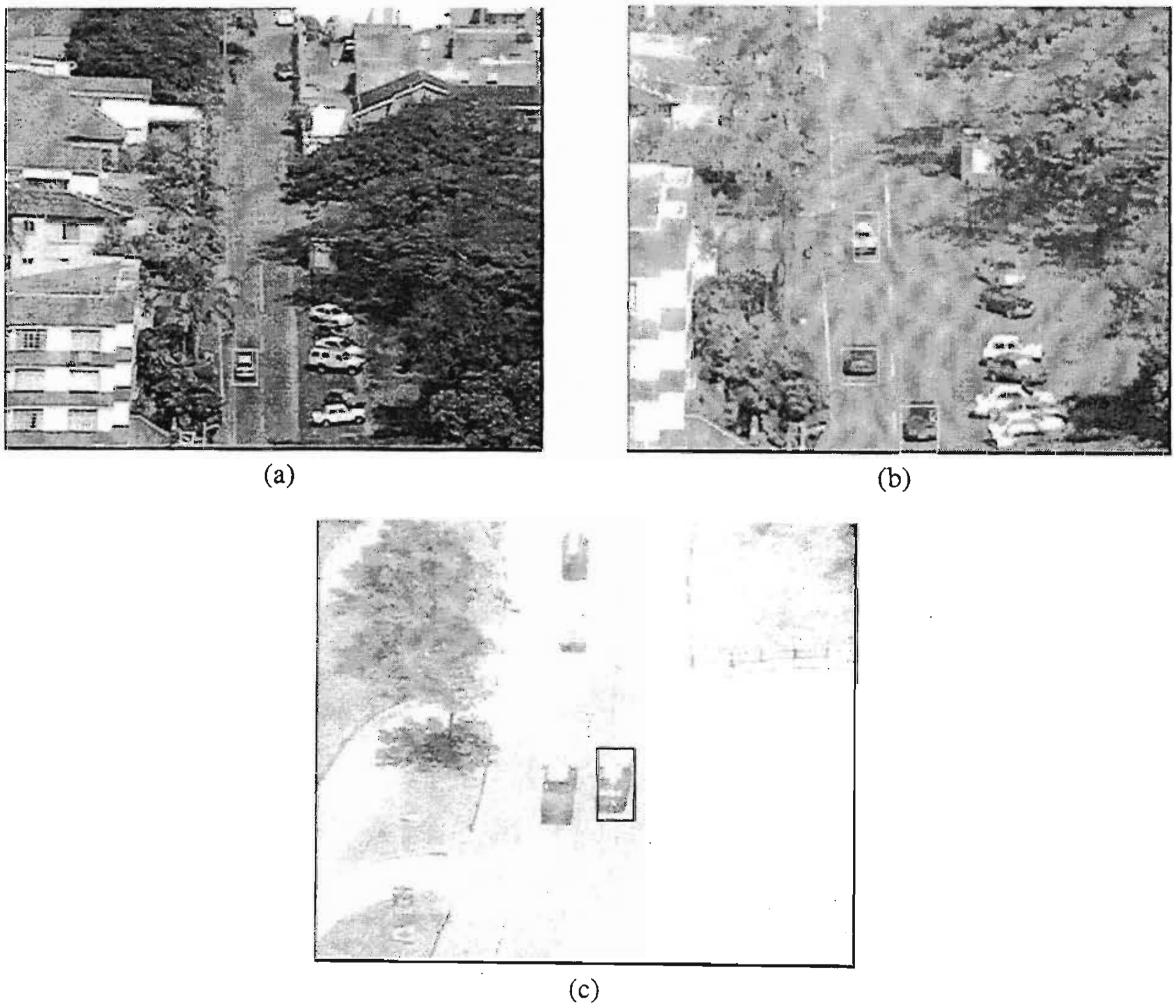


Figure 5-19: Segmentation results of varying **ClusBlock_Thres** applied to (a) VS-1, (b) VS-2 and (c) VS-3.

Figure 5-19(a) illustrates the segmentation results of VS-1 target video frame number fourteen. In this case, two objects have been segmented, the correct one being the vehicle in motion while the other at the bottom of the frame to the left of the road having resulted due to noise. Under normal circumstances however, the noise object would fall outside of the user-defined bounding area and thus would not pose a problem. **ClusBlock_Thres** values greater than 23 resulted in the noise object being filtered while values greater than 47 resulted in no objects being segmented at all.

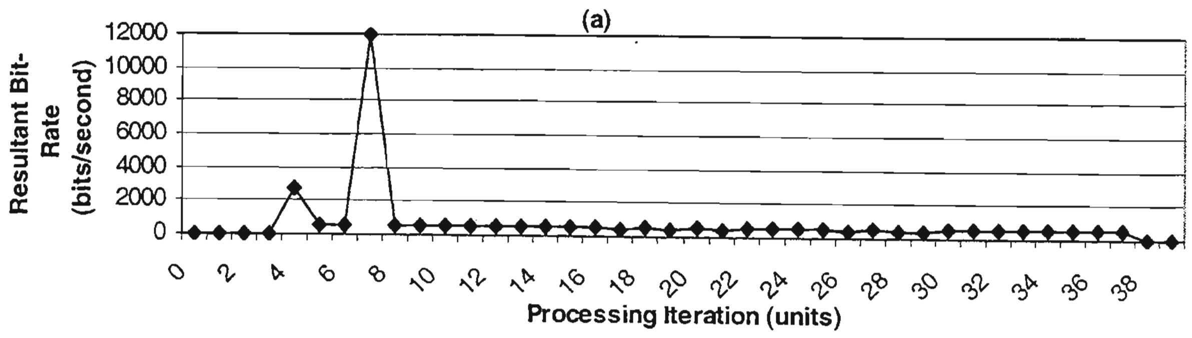
Figure 5-19(b) illustrates the segmentation results of target video frame fifteen of VS-2. In this case, the three correct objects in motion were segmented for **ClusBlock_Thres** values in the range [0, 21]. In this case however, all of the objects are not fully segmented. The partial segmentation of the white vehicle is understandable considering it is partly occluded by an overhanging tree. However the partial segmentation of the two black vehicles can only be attributed to inaccuracy. Figure 5-19(c) illustrates the segmentation results of the target video frame eight of VS-3. The vehicle in motion in the centre of the frame was correctly segmented for **ClusBlock_Thres** values in the range [0, 30].

In all three test cases, the correct objects in motion were segmented for **ClusBlock_Thres** values in the range [0, 21]. The only test frame in which noise was mistaken for a valid object in motion was VS-1. However as previously mentioned, this object would be filtered under normal circumstances as it would usually fall outside of the user-defined boundary tracking area.

5.3.3 Object Tracking Performance

The performance of the implemented tracking algorithm is evaluated according to the three test video sequences, namely VS-1, VS-2 and VS-3. In each of the three cases as far as the encoding of the reference video frame is concerned, **Thres_32** was set to ten, **Thres_16** was set twenty, **ref_matrix_size** was set to 21 and **Qfactor** was set to 100. These values were chosen in order for the decoded reference frame to be of reasonable quality for discussion purposes. With regard to the user-defined object settings, **ClusMap_Thres** was set to fifteen, **ClusBlock_Thres** was set to twenty and **ref_matrix_size** was set to 36. The user-defined object segmentation and tracking bounding area was set to include only the road area in which the objects were expected to travel. The decoder software was adapted in order to calculate and record rate metrics. In other words, following each processing iteration, the decoder would calculate rate metrics including bit-rate, YUV PSNR and compression ratio and write these values along with the elapsed encoding and decoding times to the PC's HDD. The decoder would also save each iteration's target video frame and its respective decoded video frame to HDD for discussion purposes.

Note that the results presented in this section do not consider the encoding and decoding of the reference frame. The following results pertain to the tracking of objects only.



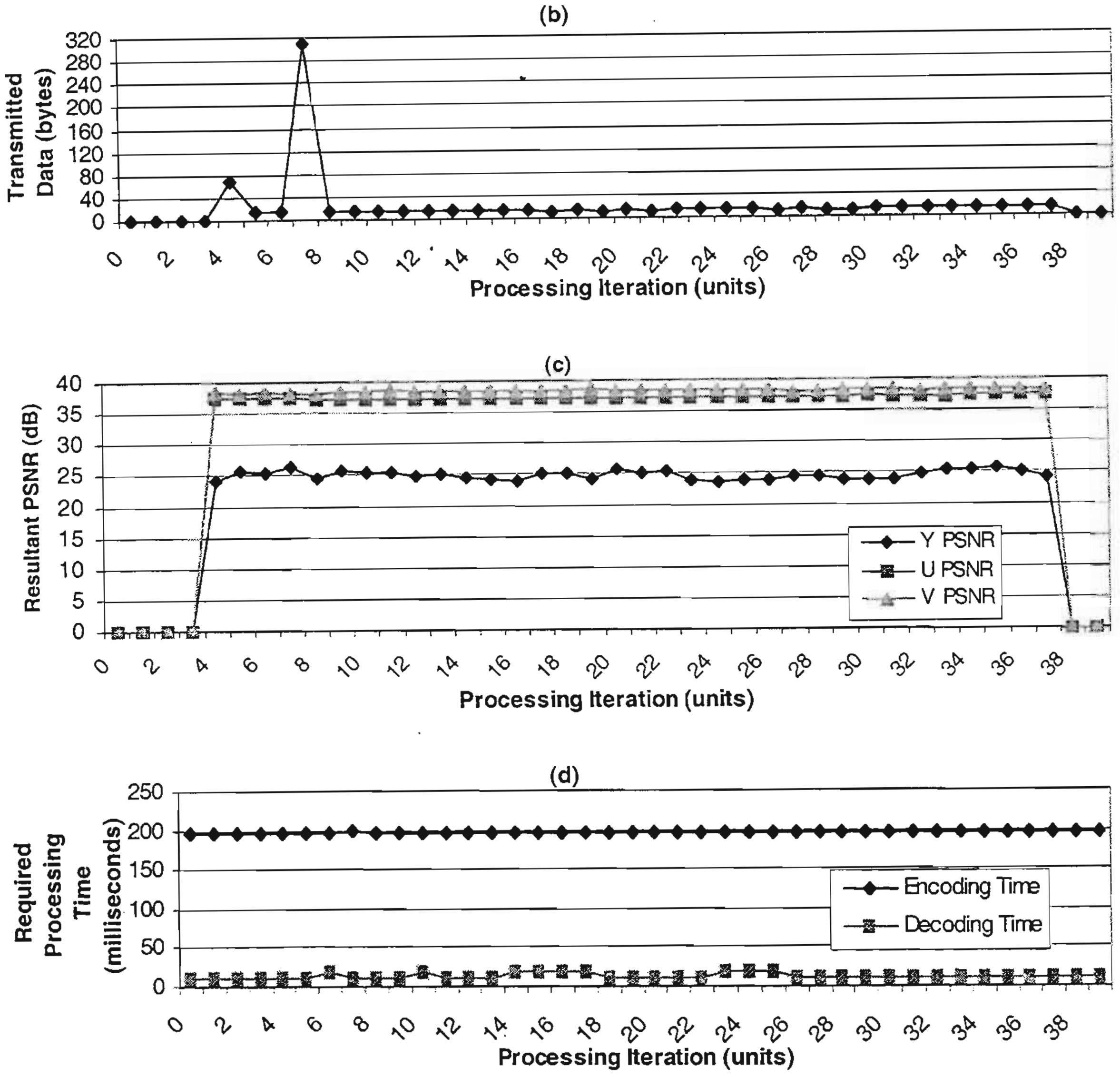


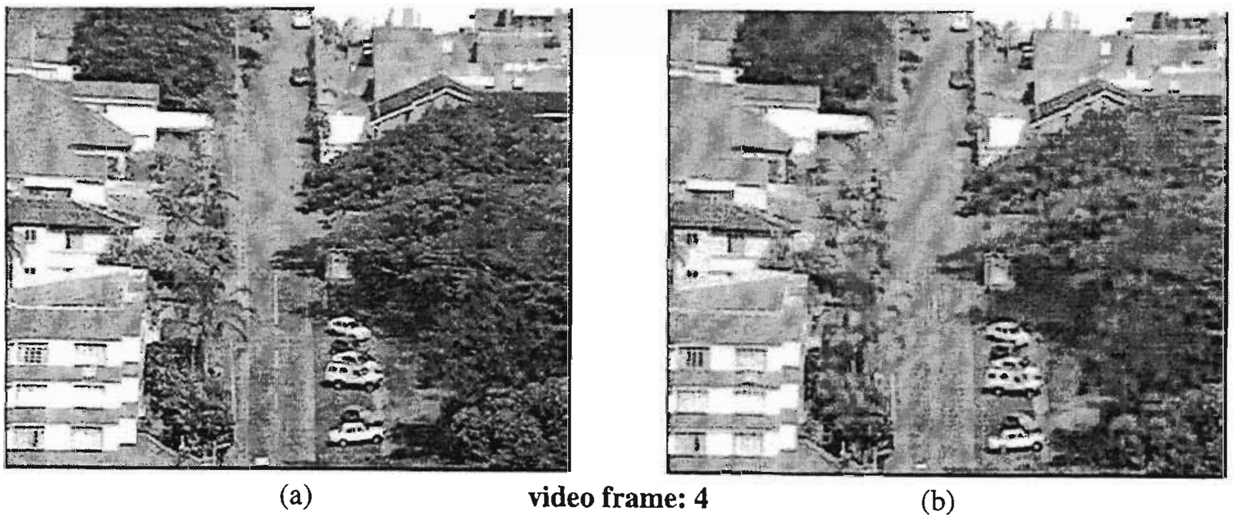
Figure 5-20: Object segmentation and tracking results of VS-1.

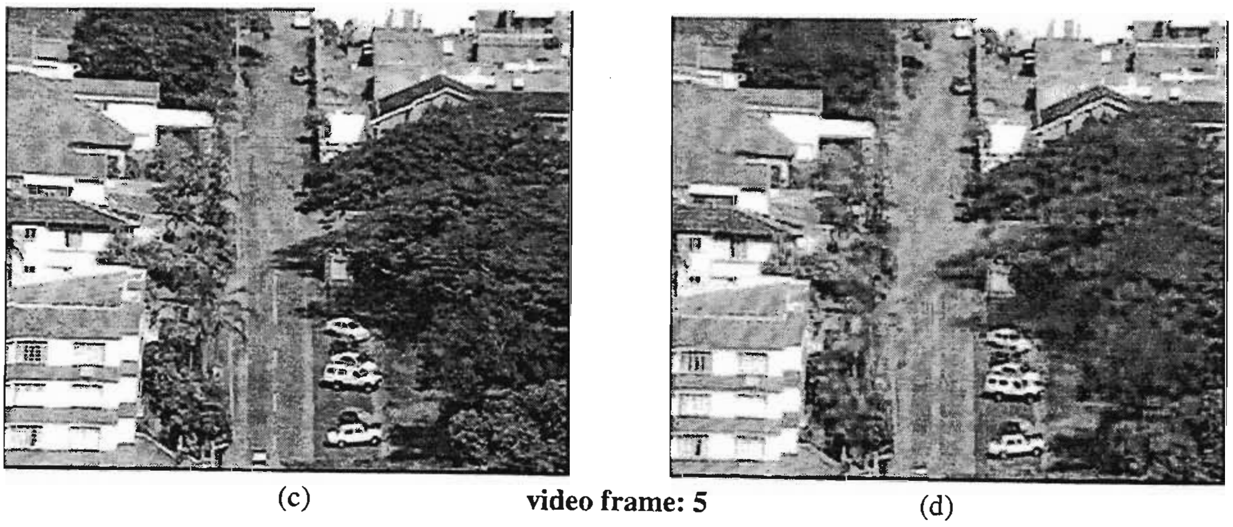
Figure 5-20 illustrates the results from the application of the implemented segmentation and object tracking algorithm to test video sequence VS-1. The results are presented as a function of the executed processing iterations. Recall from Chapter 4 that the operation of reading and writing to the shared data memory of the TriMedia and host PC is synchronous and is controlled by passing a semaphore. From this, a complete processing iteration comprises the encoder encoding a video frame or tracking objects and writing the encoded data to the shared memory space. Thereafter, the decoder reads the encoded data, decodes and displays it along with calculated rate metrics. This signals the start of the next processing iteration and so this encoding and decoding process continues over for the duration of the test video sequence.

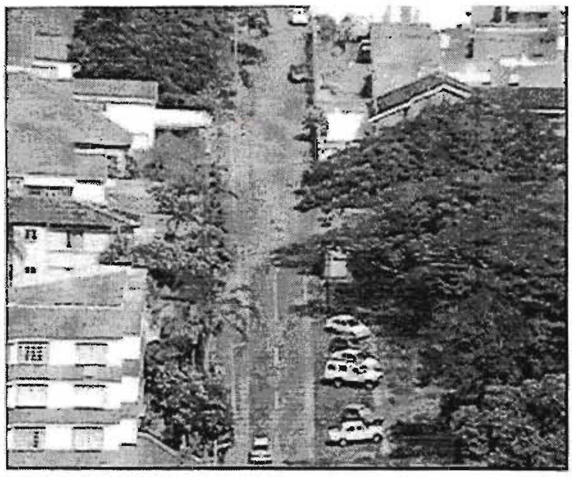
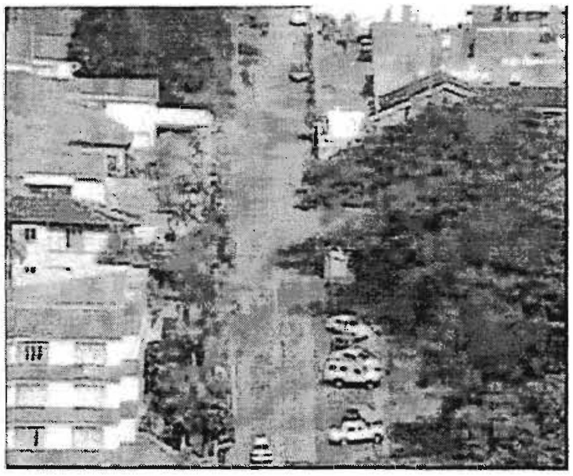
From the measured encoding and decoding times illustrated in Figure 5-20(d), a 'true' processing iteration requires approximately 206 milliseconds considering the encoding stage requires on average 196 milliseconds and the decoding stage 10 milliseconds. The word 'true' is used as the results are calculated according to this processing duration. However the time required for the calculation of the rate metrics that follow each decoding stage also need to be considered. It was measured that these calculations required on average 180 milliseconds per iteration, therefore from an evaluation point of view, a complete processing iteration required just short of 400 milliseconds. Hence the reason why VS-1, being sixteen seconds in length, only corresponds to forty processing iterations. Unfortunately the rate metric calculation duration could not be compensated for in terms of processing iterations as the tests were conducted on real-time streaming video from the video camera. However as already mentioned, the results nonetheless exclude this extra time duration. Note that the reflected encoding times include the 160 milliseconds that elapse during the capturing of the five video frames that are utilised in each processing iteration.

Figure 5-20(a) graphs the resultant bit-rate of VS-1. Note that no bit-rate limiter was implemented during the conducted object tracking evaluations in order for the *instantaneous* bit-rate to be measured. The actual amount of data in bytes that was transmitted per processing iteration is graphed in Figure 5-20(b). The YUV PSNR relationship of each decoded video frame with respect to its corresponding target video frame is graphed in Figure 5-20(c). The PSNR was only calculated when objects were tracked within the video scene hence the results are zero-valued at both the beginning and end of the graph.

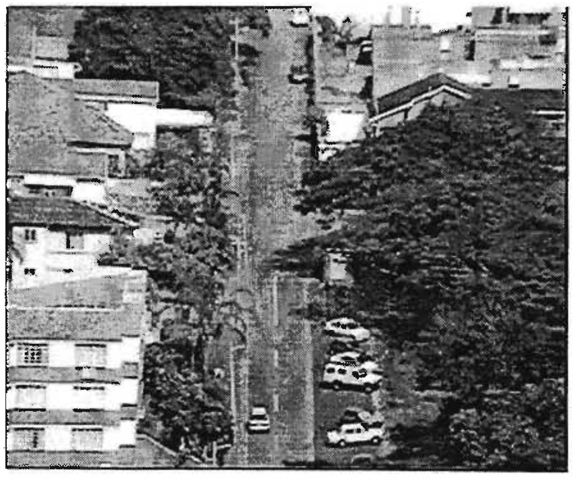
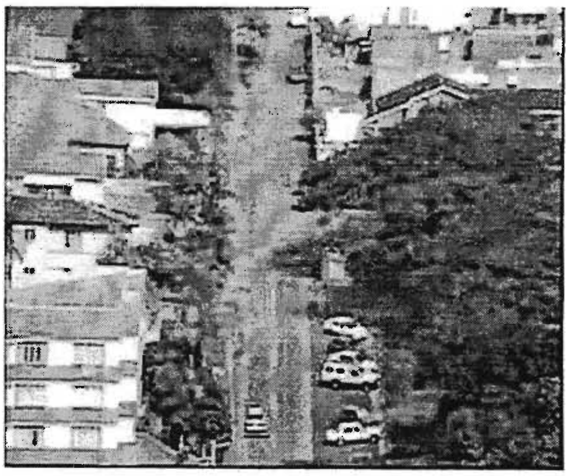
Figure 5-21 graphically compares a selected number of decoded video frames to their original target video frame counterparts. The target video frames on the left side illustrate what the decoded video frames on the right should ideally look like.


(a) **video frame: 4** (b)

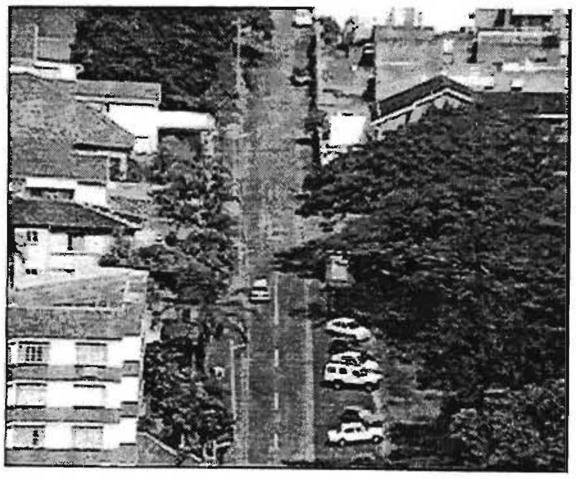
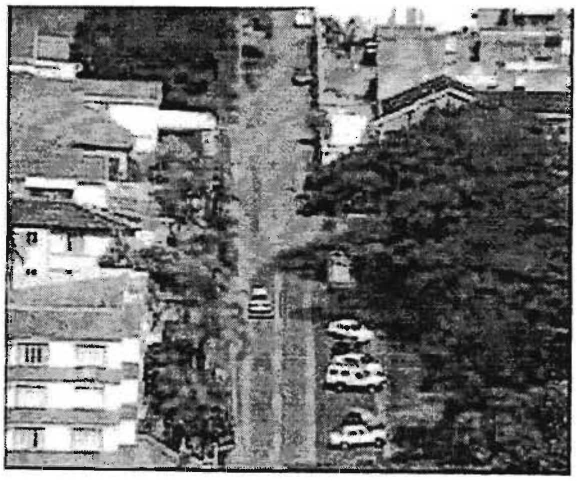

(c) **video frame: 5** (d)

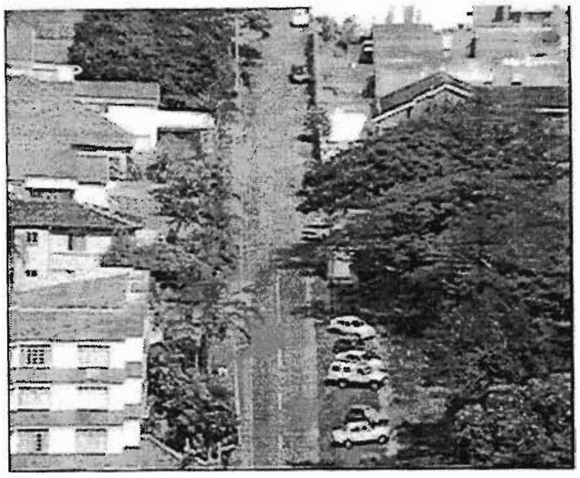
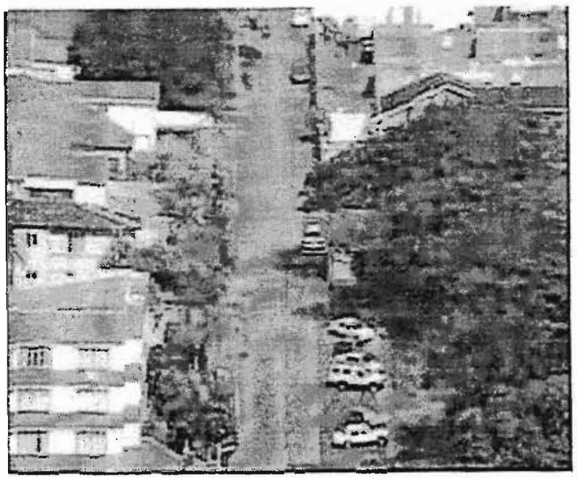
(e) **video frame: 7** (f)

(g) **video frame: 11** (h)

(i) **video frame: 24** (j)

(k) **video frame: 34** (l)

Figure 5-21: Examples of VS-1 with the target video frames to the left of their decoded counterparts.

From the resultant bit-rate graphed in Figure 5-20(a) and the actual number of bytes transmitted graphed in Figure 5-20(b), no data is transmitted to the decoder up to and including processing iteration three as no objects were detected. A vehicle entering the bottom of the video scene is detected in processing iteration number four resulting in 63 bytes of encoded data being transmitted to the decoder. The corresponding decoded video frame is illustrated in Figure 5-21(b). During iterations five and six, the algorithm tracks the buffered object hence the low bit-rate of 507 bits/second. The tracked object in processing iteration five is illustrated in Figure 5-21(d).

In iteration seven, the tracking algorithm detects that the object has increased in size thus its buffered YUV components are updated, encoded and transmitted to the decoder. At this point, the resulting bit-rate is calculated at 11,961 bits/second while the actual amount of object data transmitted is 311 bytes. The segmented object was thereafter consistently tracked from iterations eight to thirty and as a result the bit-rate never exceeded 505 bits/second. Examples of decoded video frames during this tracking sequence are illustrated in Figures 5-21(h) and (j).

From processing iteration number 31 onwards the object starts moving around the corner at the top of the road. Recall that the segmentation and tracking algorithm was not specifically designed to cater for objects that rotate in motion. The consequence of this is noted on comparing Figures 5-21(k) and (l). The algorithm continues to track the vehicle even though is has actually rotated. However in this case, it superimposes the buffered version of the vehicle onto the reference frame even though it no longer resembles the correct shape of the current vehicle. Although the decoded video frame illustrated in Figure 5-21(l) may appear aesthetically displeasing, it still conveys enough information in that the motion of the object is still understandable. The object was no longer in the video scene in processing iterations 38 and 39 hence no data was transmitted to the decoder.

Relating the vehicle's size and the average number of pixels that it travelled between processing iterations to equation 4-7, it is estimated that the vehicle was travelling at approximately 30 km/h. It is further estimated that a vehicle this size would still be successfully tracked, considering the 24 pixel motion tracking area, if it travelled at any speed up to a maximum of approximately 90 km/h.

With regard to the environment depicted in VS-1, the results indicate that the implemented algorithm is able to successfully segment and track a reasonably small, slow moving object that changes in size and moves mostly with translational motion. Furthermore the algorithm has the ability to tolerate rotational motion even if the resultant decoded video frame does not represent the true appearance of the object.

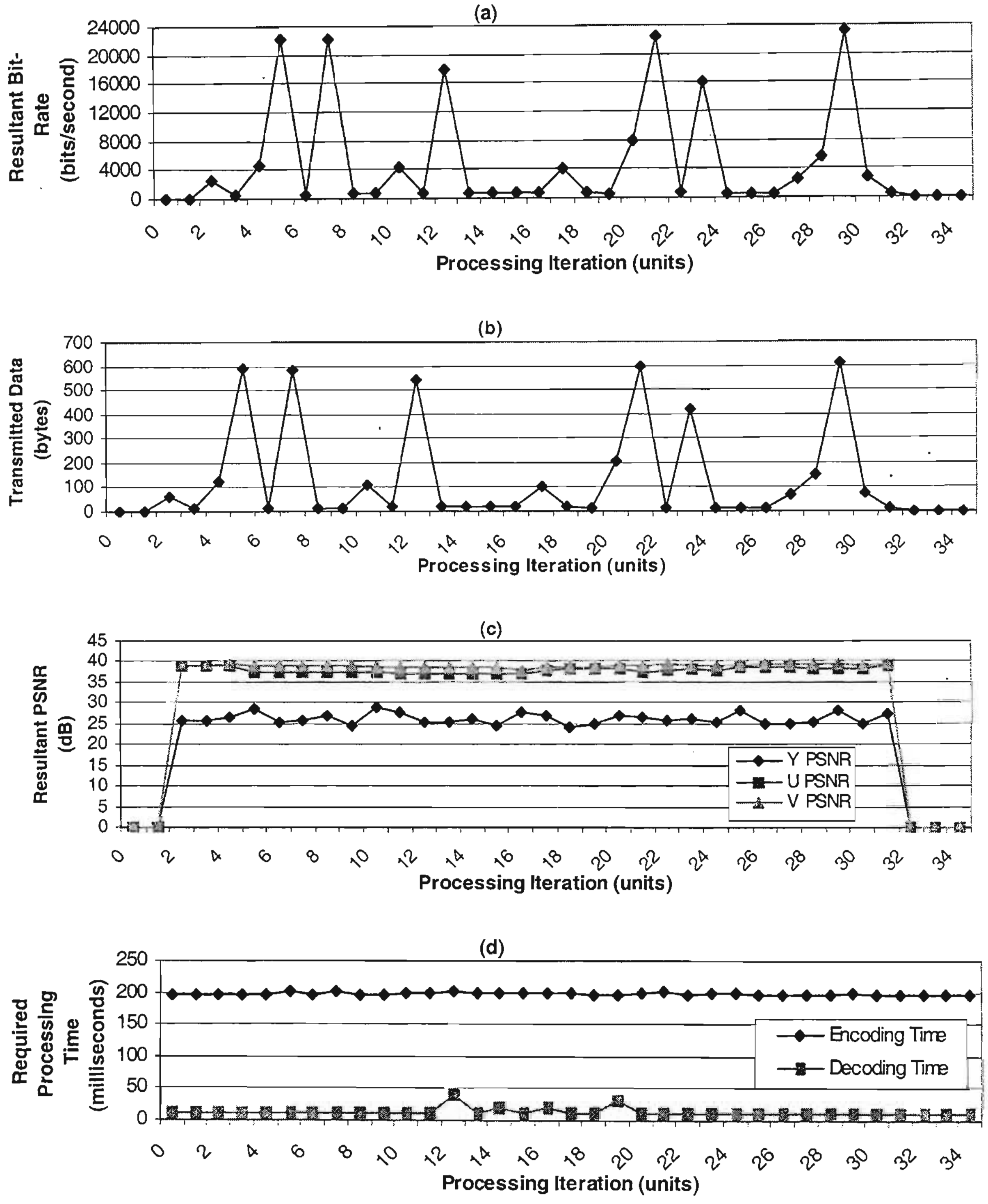
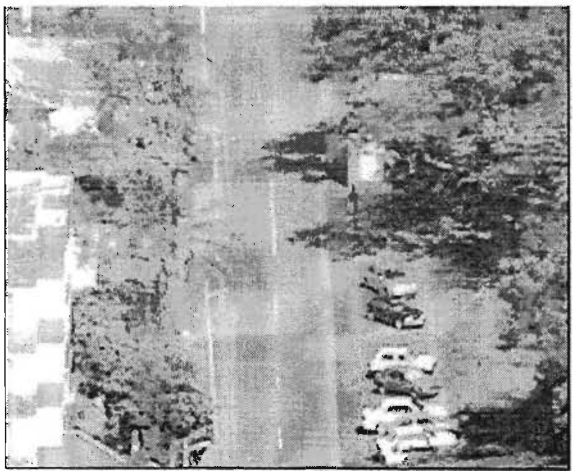
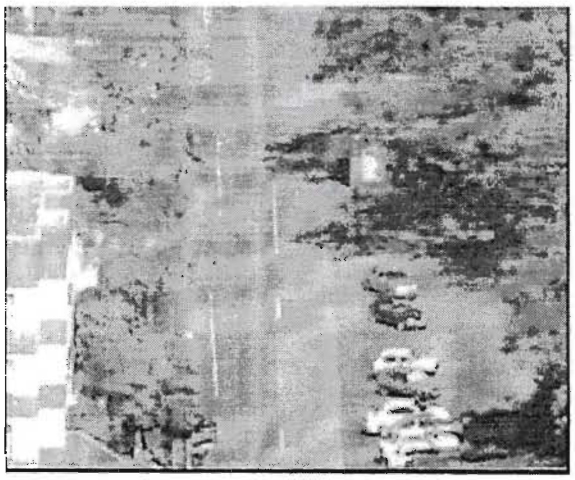
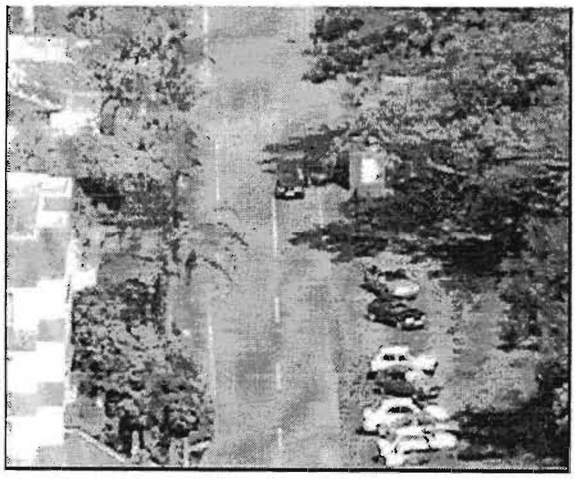
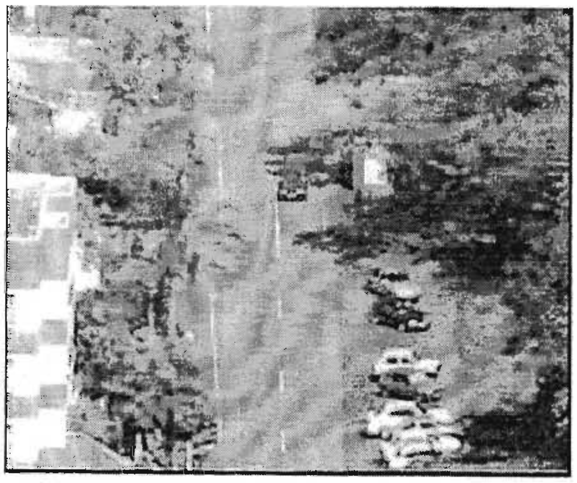


Figure 5-22: Object segmentation and tracking results of VS-2.

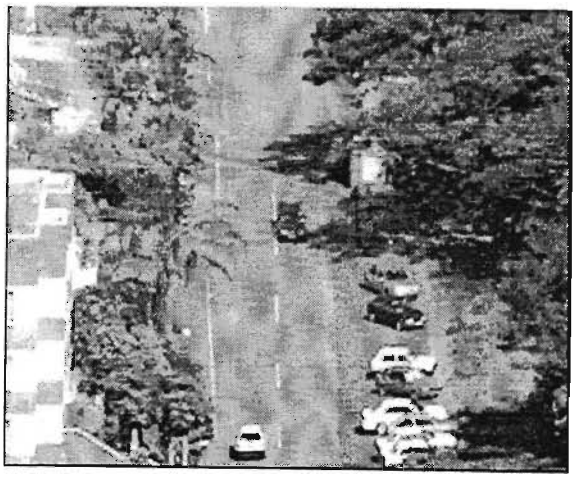
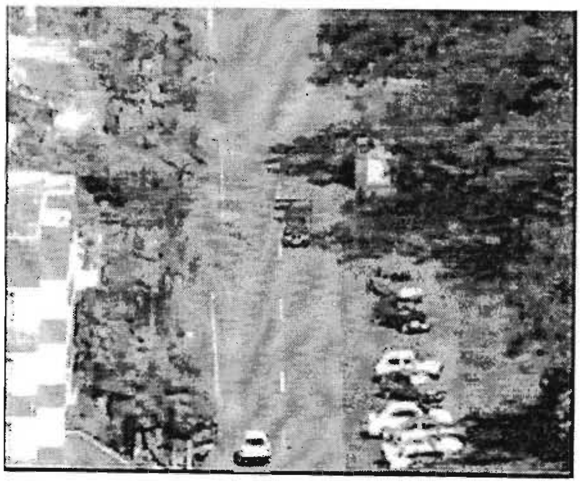
Figure 5-22 illustrates the results that were obtained from the application of the implemented algorithm to test video sequence VS-2. VS-2 is fourteen seconds in length and therefore corresponds to approximately 35 processing iterations. From the timing results presented in Figure 5-22(d), the encoding time per iteration is once again consistently less than 200 milliseconds while the decoding average is approximately ten milliseconds. Example frames of the decoded sequence are illustrated in Figure 5-23. Once again the target video frames are to the left of their decoded counterparts.

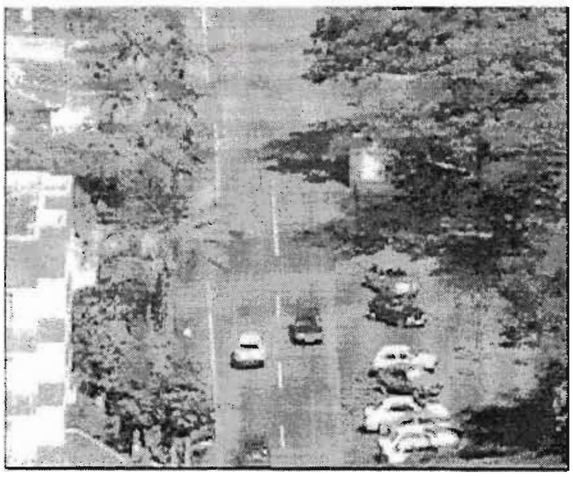
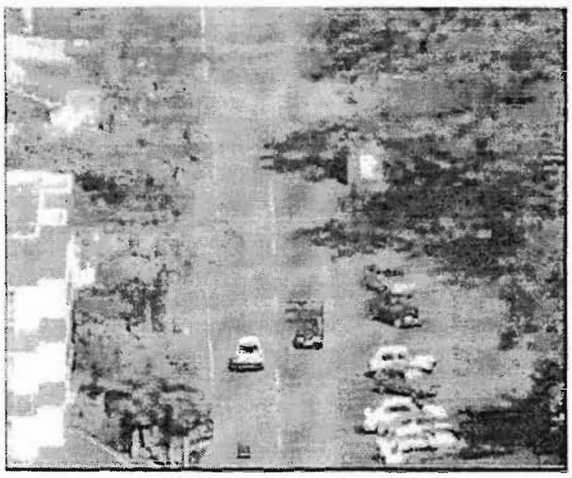
(a) **video frame: 2** (b)

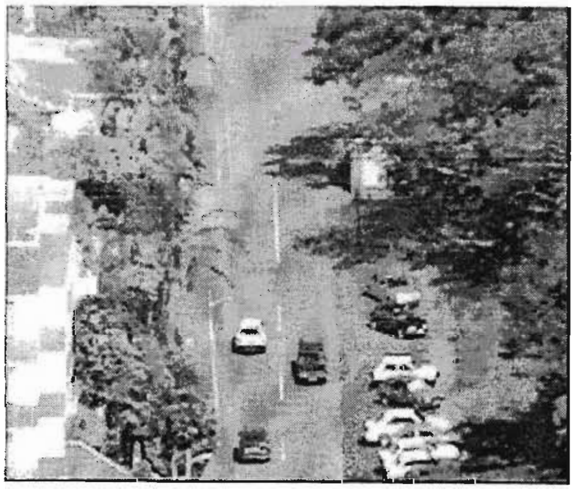
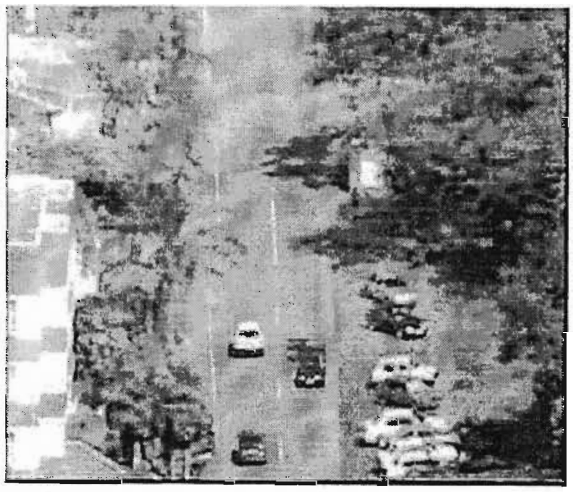
(c) **video frame: 5** (d)

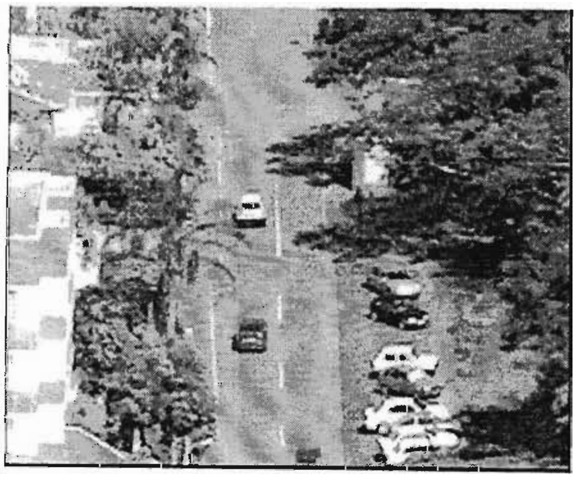
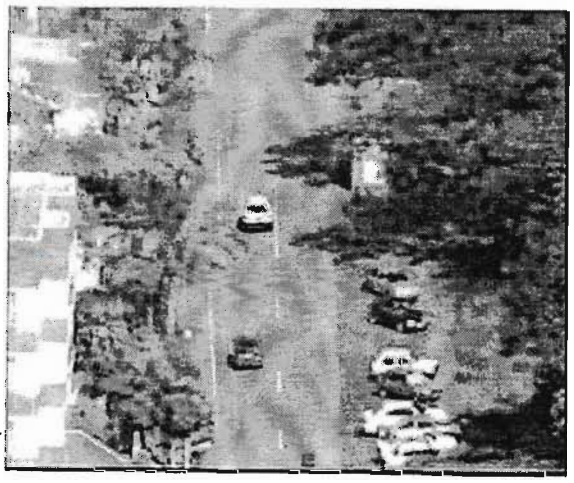
(e) **video frame: 7** (f)

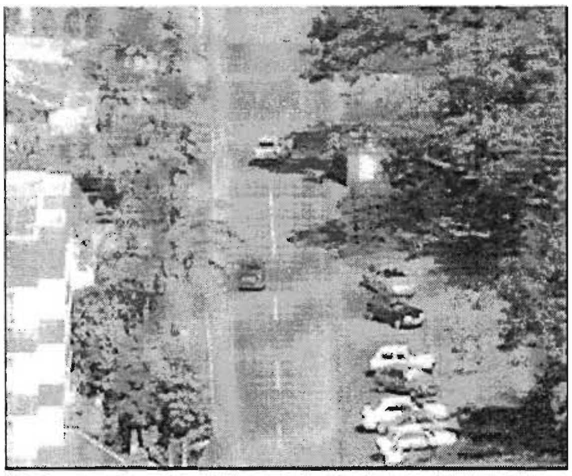
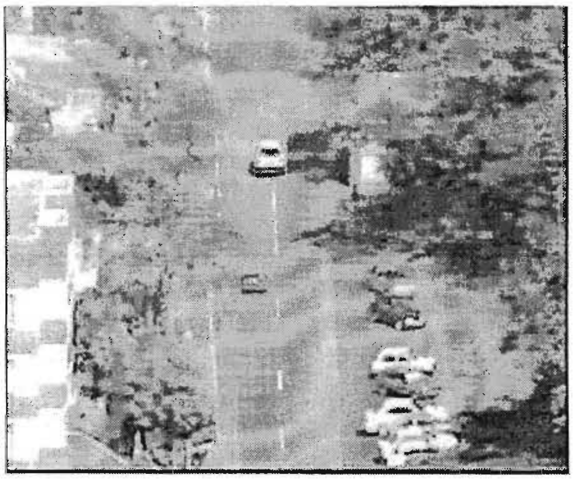
(g) **video frame: 11** (h)

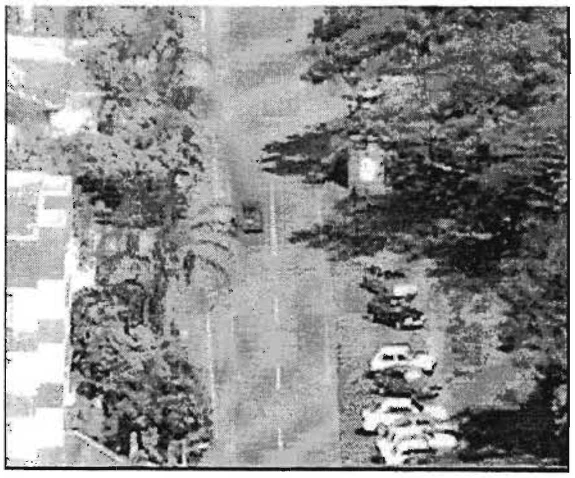
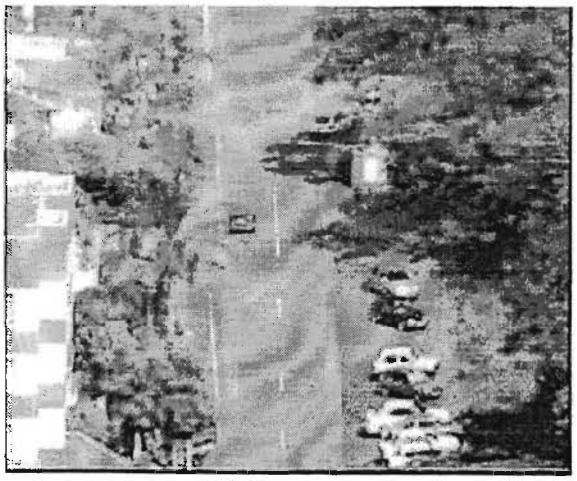
(i) **video frame: 12** (j)

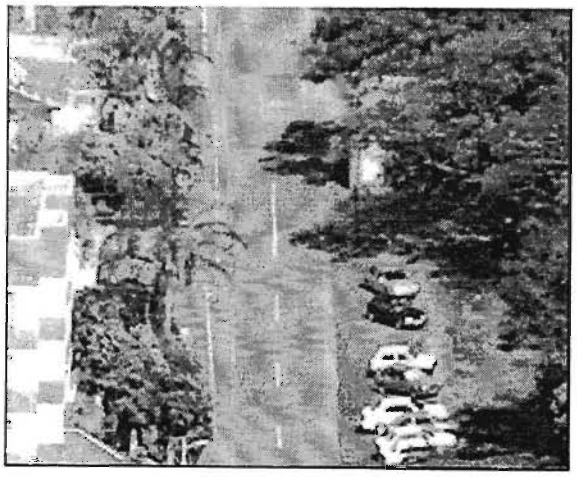
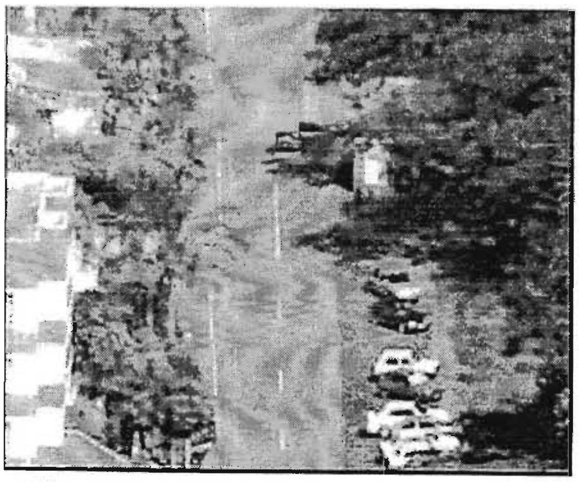
(k) **video frame: 17** (l)

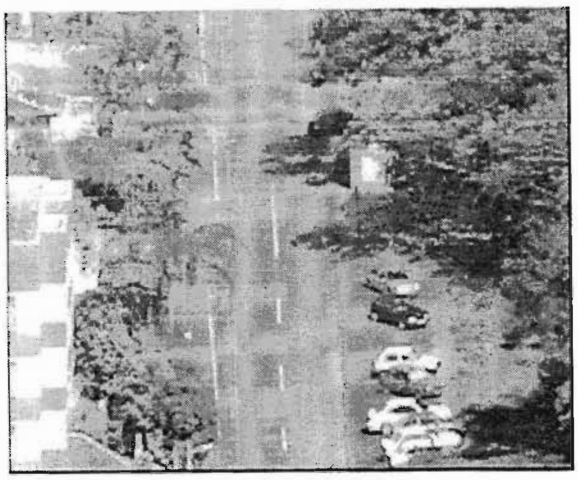
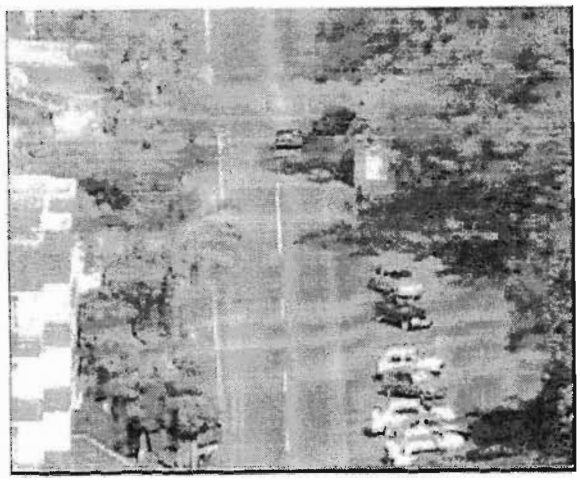
(m) **video frame: 20** (n)

(o) **video frame: 23** (p)

(q) **video frame: 28** (r)

(s) **video frame: 29** (t)

Figure 5-23: Examples of VS-2 with the target video frames to the left of their decoded counterparts.

In the first iteration no motion was detected and as a consequence no object data was transmitted. In the third iteration however, the algorithm detected the black vehicle moving around the corner towards the top of the road and as a result, 65 bytes of encoded object data were transmitted to the decoder.

As the moving object rounds the corner, the algorithm detects that it has increased in size hence 122 bytes of updated object data are transmitted in processing iteration four. The object rounds the corner completely in iteration five and once again, updated object data is transmitted to the decoder. The measured instantaneous bit-rate as a result of the increase in size was measured at 22,255 bits/second. It is noted that the decoded video frame illustrated in Figure 5-23(d) is a reasonably accurate representation of the original input illustrated in Figure 5-23(c).

In processing iteration seven, the white vehicle towards the bottom of the video scene was detected and its segmentation resulted in 584 bytes being transmitted to the decoder. It must be mentioned however that the algorithm missed the vehicle as it first entered the scene and only once it was fully in the scene was it detected. In the decoded frame illustrated in Figure 5-23(f), the back of the segmented black vehicle appears slightly distorted. This distortion is actually the shadow of the overhanging tree and was unfortunately included as the algorithm updated the segmented object. The two objects are successfully tracked in iterations eight and nine while a further object entering the scene was detected in iteration ten. Figure 5-23(g) illustrates the target video frame that resulted from processing iteration number eleven. In this instance, the tracking algorithm was unable to detect that the new object had increased in size to the point where it was almost fully visible in the video scene. This result is depicted in Figure 5-23(h). A possible reason for this inaccuracy is that the cluster block thresholds were set too high hence filtering the edges of the newly detected object. Only once its size was definite in iteration twelve was it updated resulting in 544 bytes of data being transmitted to the decoder.

The three objects were tracked successfully during iterations thirteen to sixteen with each iteration resulting in only twenty bytes being transmitted to the decoder. The slight increase in bit-rate in iteration seventeen resulted due to the black vehicle moving out of the scene hence reducing in size. Once again however, possibly due to the filtering thresholds being too high, the vehicle is not segmented correctly and is depicted by a column of black blocks.

The remaining two objects are tracked successfully during processing iterations eighteen and nineteen, however in processing iteration twenty, from the target video frame depicted in Figure 5-23(m), the black vehicle has moved behind the branches of an overhanging tree. Although the tree appears insignificant in terms of size, the slight occlusion of the vehicle resulted in the object being updated by the algorithm. When considering the white vehicle as it moved underneath the same tree, the algorithm did not exhibit a similar partial occlusion problem. This result possibly suggests that the tracking algorithm is able to track vehicles that are brighter in colour better than vehicles that are dark or dull in colour. This would be understandable as a white vehicle on a dark road is higher in contrast than a dark vehicle on the same road. In this instance, the edges of the white vehicle would be more defined and would thus aid tracking whereas the edges of the dark vehicle could possibly be confused with the dark road hence detract from accurate tracking.

In processing iteration 23, the increase in output bit-rate is a result of three objects being tracked within the video scene even though there are actually only two legitimate objects. From the decoded frame illustrated in Figure 5-23(p), the algorithm has detected the white vehicle moving behind a tree towards the top of the road while at the same time, a previous version of the buffered vehicle is also superimposed behind the current correct version. Therefore in this iteration, two instances of the white vehicle are actually being tracked. Recall the algorithm was designed such that if an object is not successfully tracked after three consecutive iterations, it is then deleted from memory. An example of this execution is the white vehicle. From the transmitted data graph in Figure 5-22(b), the second instance of the white vehicle was first tracked in iteration 21. The algorithm eventually determined that the old version was no longer present within the scene hence it was deleted from memory in iteration 24.

Similarly in iterations 27 and 28, the algorithm mistakenly tracked the black object moving around the same corner in the shadow as two separate objects. In this case, the front and back ends of the vehicle have been segmented independently of each other as illustrated in Figure 5-23(r). In iteration 29, the algorithm segments the vehicle in its correct position within the scene but at the same time superimposes the old version behind it. The old version is eventually deleted in iteration 31 and from iteration 32 onwards, the current version of the object is not tracked as well.

Relating the measured sizes of the three vehicles and the number of pixels they moved between processing iterations to (4-7), it is estimated that the vehicle on the right was travelling at approximately 48 km/h and the two on the left at approximately 43 km/h. It is further estimated that these vehicles would still be successfully tracked if they travelled at any speed below 65 km/h.

To summarise the performance of the system with regard to the environment depicted in VS-2, the results indicate that the algorithm is able to successfully track medium size vehicles travelling at moderate speeds. Some weak points of the algorithm that were identified include inconsistency in terms of segmenting the complete visible portion of a vehicle as it enters or exits a video scene as well as inaccuracy as a vehicle moves within the shadow of an overhanging tree for example. As a consequence of this inaccuracy, multiple instances of the same object could result which is aesthetically displeasing. Some positive aspects that were identified include ability of the algorithm to track multiple objects simultaneously and it's ability to correct itself in terms of tracking multiple instances of the same object. Overall, the quality and accuracy of the decoded video sequence is felt to be of a reasonable standard.

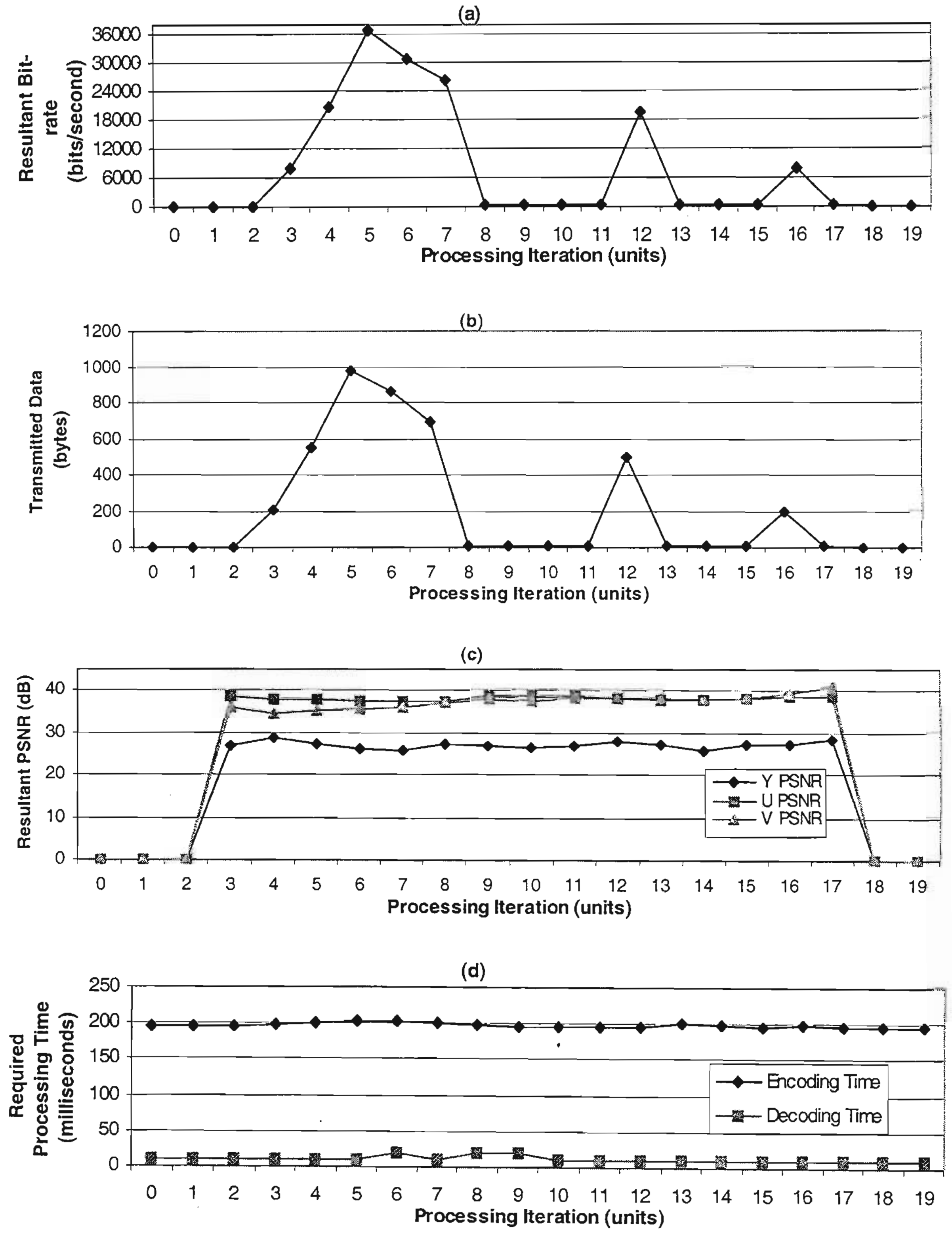
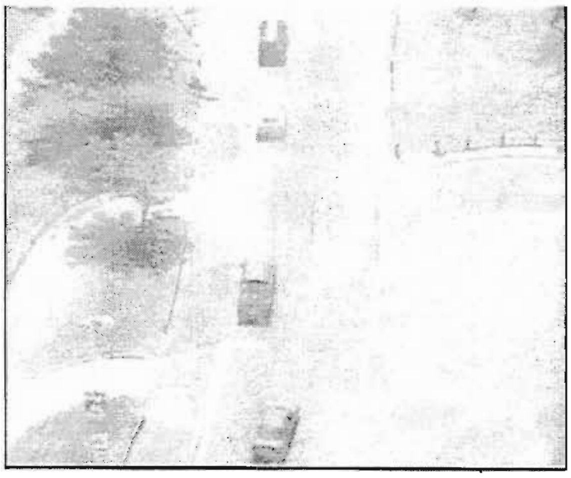
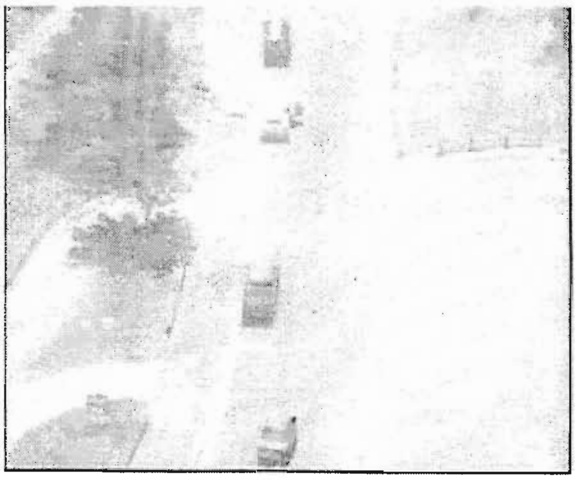
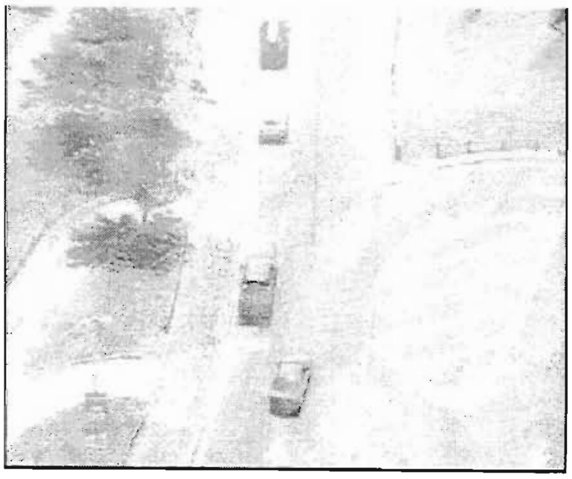
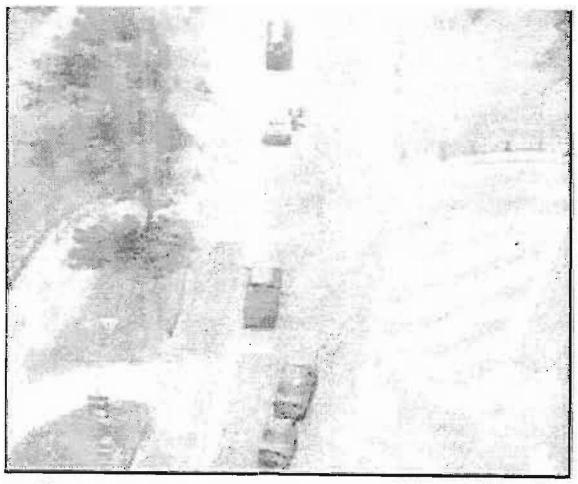


Figure 5-24: Object segmentation and tracking results of VS-3.

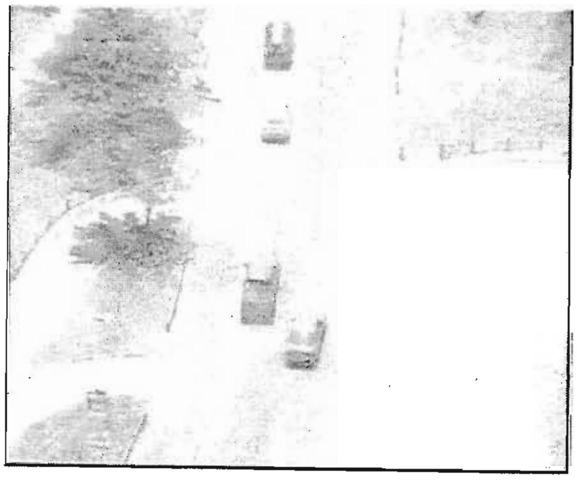
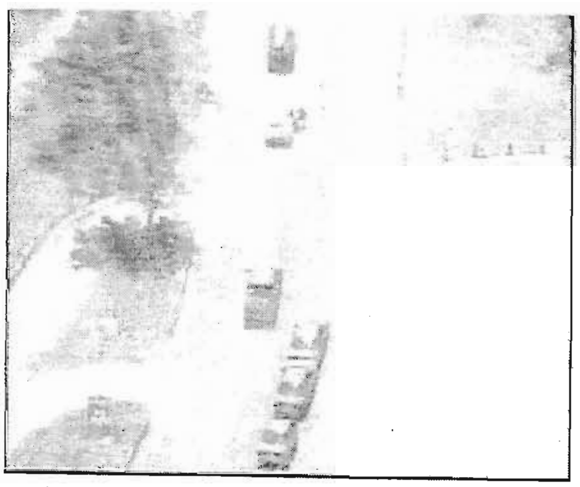
Figure 5-24 illustrates the results that were obtained from the application of the object segmentation and tracking algorithm to test video sequence VS-3. VS-3 is approximately eight seconds in duration and corresponds to approximately twenty processing iterations. The instantaneous calculated YUV PSNR values of the decoded video sequence is graphed in Figure 5-24(c) and required encoding and decoding times are illustrated in Figure 5-24(d). The resultant bit-rates and transmitted data graphs are discussed in greater detail with respect to the decoded video frame examples illustrated in Figure 5-25. Once again the video frame on the left illustrates what the decoded frame on the right should ideally look like.

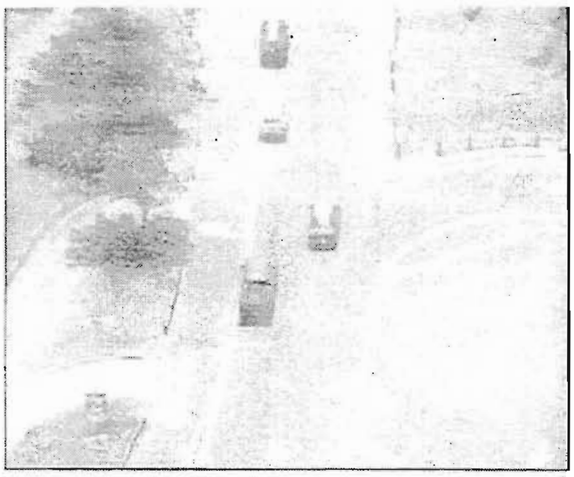
(a) **video frame: 4** (b)

(c) **video frame: 5** (d)

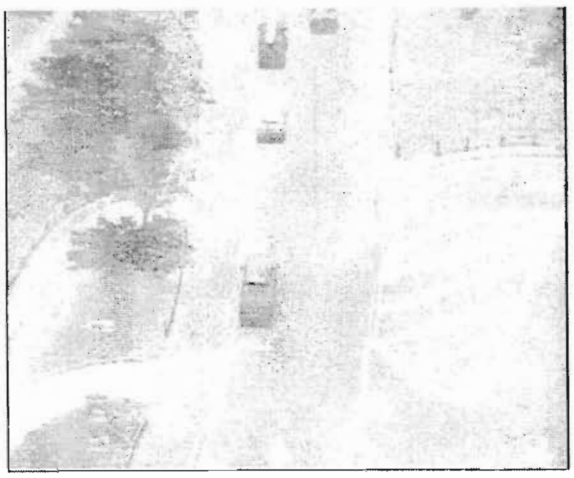
(e) **video frame: 6** (f)

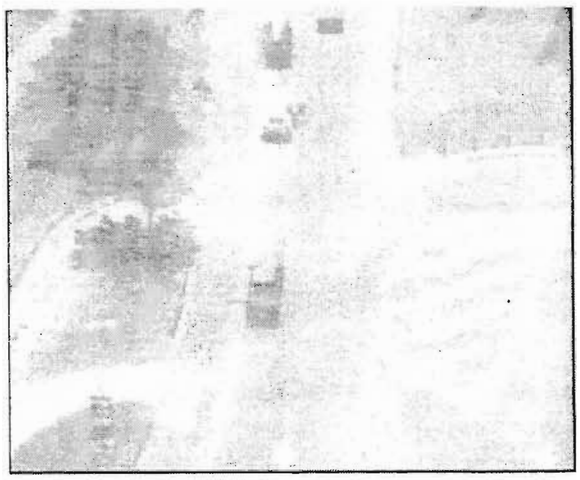


(g) **video frame: 9** (h)

(i) **video frame: 13** (j)

(k) **video frame: 15** (l)





(m) **video frame: 16** (n)

Figure 5-25: Examples of VS-3 with the target video frames to the left of their decoded counterparts.

With respect to the transmitted data graph illustrated in Figure 5-24(b), no objects were segmented or tracked within the first three processing iterations hence no data was transmitted. In processing iteration three, the algorithm detected an object entering the bottom of the video scene and as a result 203 bytes were transmitted to the decoder. The resultant bit-rate increases in iteration four due to the algorithm detecting the object's increase in size. The decoded frame is illustrated in Figure 5-24(b).

The largest amount of data throughout the test sequence was transmitted in processing iteration five. The cause of this, as illustrated in Figure 5-25(d), is that the algorithm has incorrectly tracked the moving object and as a consequence has superimposed two instances of the same object onto the reference frame. The situation is compounded further in iteration six when the algorithm once again incorrectly tracks the object and ultimately ends up tracking three instances of the same object. This erroneous segmentation and tracking suggests that the object is possibly moving too fast for the algorithm to successfully track it. The first erroneous instance of the object is deleted from memory in iteration seven and the second instance is deleted in iteration eight. From processing iteration eight to eleven, only one instance of the object is tracked correctly.

The algorithm makes a similar mistake in processing iteration twelve in that two instances of the same object are tracked thus resulting in a significant increase in output bit-rate. In iteration fifteen, the algorithm corrects itself by deleting the old version of the tracked object. 200 bytes are transmitted in iteration sixteen due to the object reducing in size as it starts to exit the scene. No further object data is transmitted from iteration nineteen onwards.

The poor tracking performance of the algorithm in this particular environment can be explained by analysing the vehicle's movement in greater detail. It is estimated that in this instance, the vehicle was travelling at approximately 59 km/h. With regard to the graphical analysis of the relationship between a vehicle's size and speed illustrated in the previous chapter in Figure 4-25, it is estimated that the maximum speed a vehicle of this size can travel at is approximately 60 km/h. Recall that for the purposes of evaluation, the motion tracking area was defined at 24 pixels. This tracking area was calculated in order for the algorithm to track reasonable sized vehicles travelling at speeds between 20 km/h and 60 km/h. The results clearly indicate that this tracking area is not suitable for larger sized vehicles travelling at speeds within the intended range. This reiterates the tradeoff of the implemented algorithm in terms of the size of a vehicle and the maximum speed that it can travel at in order for it to be successfully tracked. A suitable improvement to the system would be to allow the motion tracking area to be user-defined.

5.3.4 Ability to Track Halted Objects

Recall from Chapter 4 that an important factor that was considered during the design of the tracking algorithm was its ability to track objects that have halted in their paths. This feature was necessary considering the tracking algorithm tracks the motion of an object rather than the object itself. Assuming the encoder was not designed to track halted objects, if a tracked object had to suddenly stop moving, the algorithm would assume that the object was no longer visible within the video scene due to it not generating any motion. The lack of motion would result in the object being deleted from memory and thus it would not be included in the resultant decoded frames. Only if the object had to start moving again would it be re-segmented and tracked. The net result of objects disappearing and reappearing would be unacceptable from anyone's point of view thus the algorithm was designed to cater for such situations.

The algorithm was designed so that in the event of a tracked object suddenly stopping, after three consecutive processing iterations in which the object is not tracked, a diamond zonal search is executed. Its purpose is to determine if the object, or at least a part of it is still present within the video scene. As a final check on completion of the search, in order to determine if a possibly located object indeed matches its buffered counterpart, the mean values of the two objects are calculated and compared. The system operator defines a % Mean Difference threshold that determines the maximum range that the mean values can differ from each other in the form of a percentage. The threshold has a range of [0.5, 5.0].

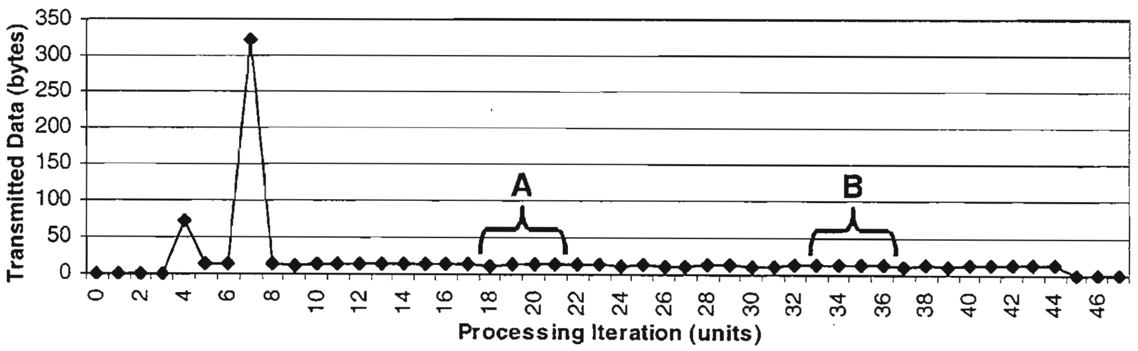


Figure 5-26: Resultant transmitted data of VS-1 when the streaming input video was paused.

The algorithm's ability to track halted objects was tested in a similar fashion to the evaluation procedure conducted in the previous section. Video from the target video camera was streamed into the TriMedia and encoded. However in this case, at certain points within the video sequence, the pause button on the video camera was pushed therefore simulating the action of a moving object suddenly stopping. If the algorithm was able to successfully track a halted vehicle, it would remain present within the decoded output throughout the paused duration. Figure 5-26 illustrates the transmitted data graph of test video sequence VS-1 that resulted when the streaming input video was paused on two separate occasions. The sequence was paused at processing iterations 18 and 33. In both cases, the video sequences were paused for the equivalent of four processing iterations. The iterations that were executed within these paused periods are indicated in Figure 5-26 by the brackets labelled A and B respectively.

In both cases, there was no increase or decrease in the amount of transmitted data both during and after the paused periods. This provides an indication that the halted object was tracked successfully. During the paused period labelled A, the % mean difference was recorded at 4.98% and during period B, the % mean difference was recorded at 4.28%. Figure 5-27 illustrates the resulting decoded video frames for (a) processing iteration eighteen, (b) processing iteration 21, (c) processing iteration 33 and (d) processing iteration 36.

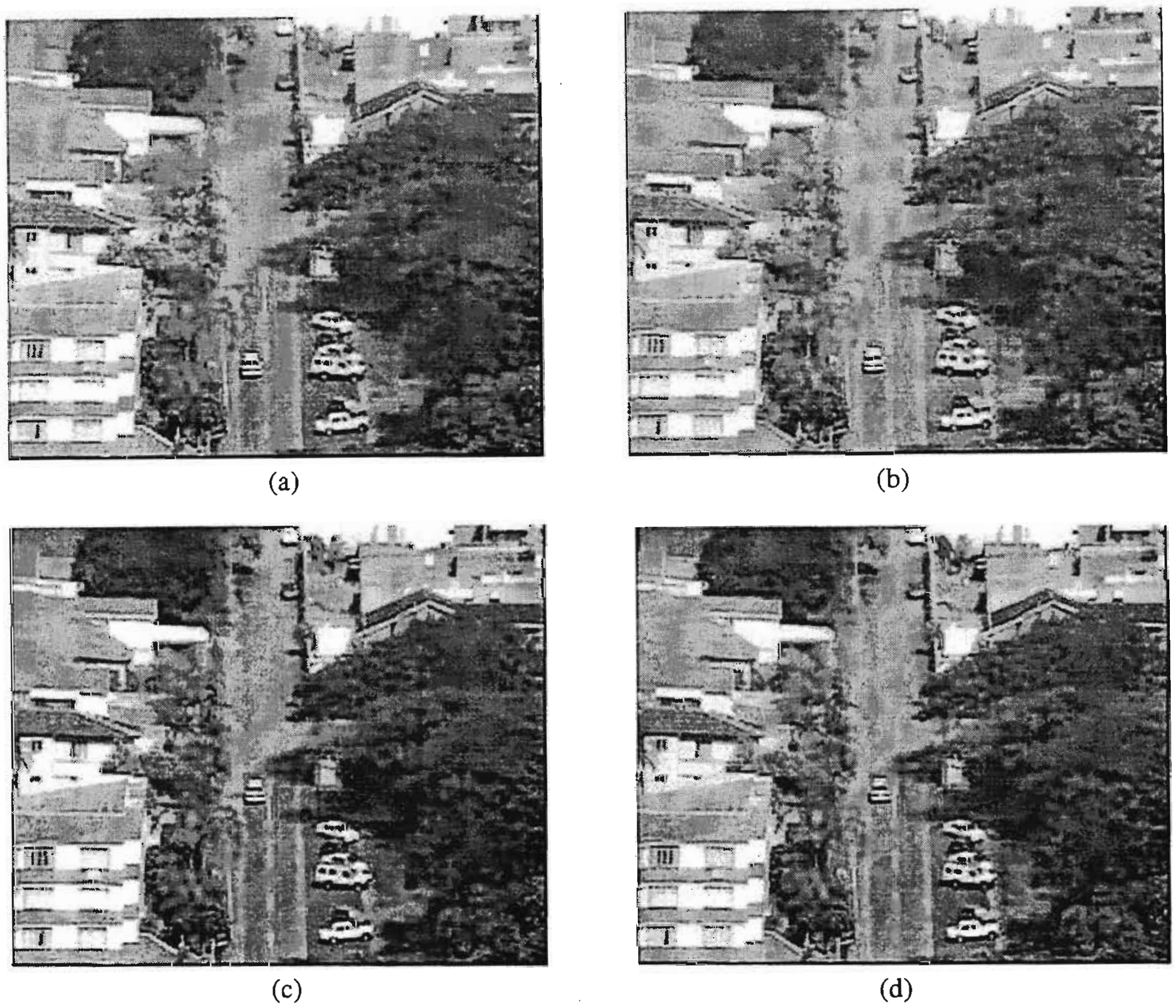


Figure 5-27: Resultant decoded video frames of a tracked halted object in VS-1.

Figure 5-28 illustrates the resultant transmitted data to the decoder during the evaluation of VS-2. In this sequence, the streaming video was once again paused in two separate places. In this case, it was paused during iterations 12 to 27 represented by A in Figure 5-28, and iterations 39 to 49 represented by B. As in the previous set of results, there was no increase in the amount of transmitted data thus indicating the successful tracking of the halted object. The % mean difference during the paused period A was recorded at 4.08% and 2.66 % during period B. Obviously the lower the recorded mean percentage difference, the more successful the search algorithm was in matching the buffered object to its halted counterpart within each target video frame.

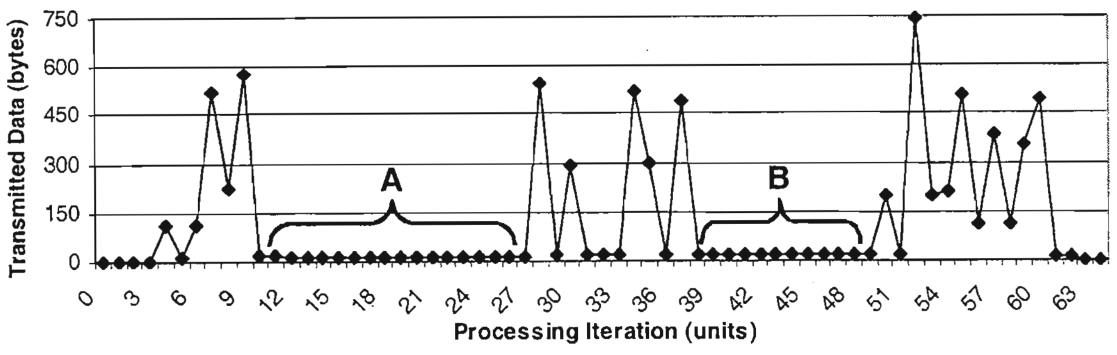
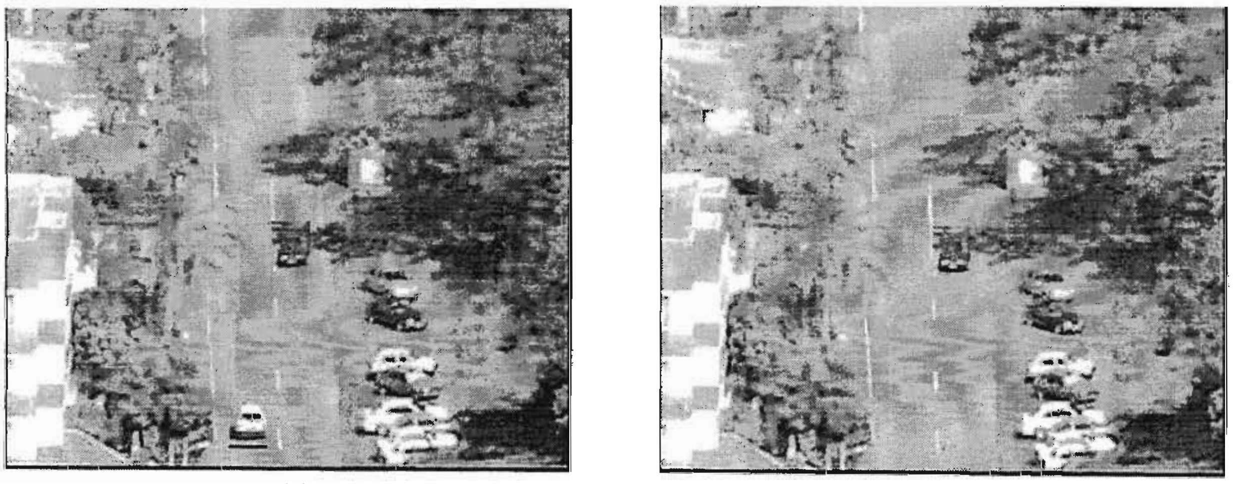
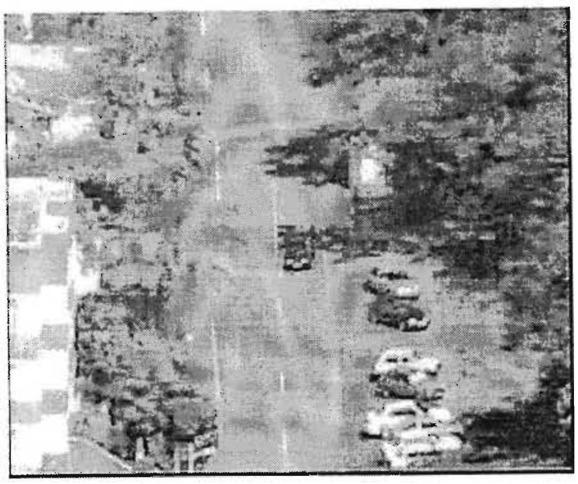


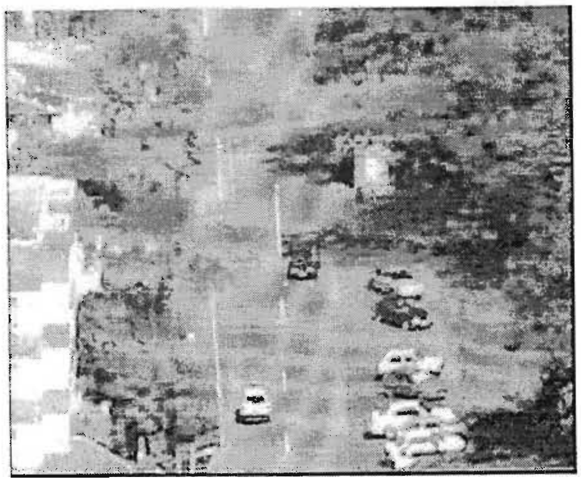
Figure 5-28: Resultant transmitted data of VS-2 when the streaming input video was paused.

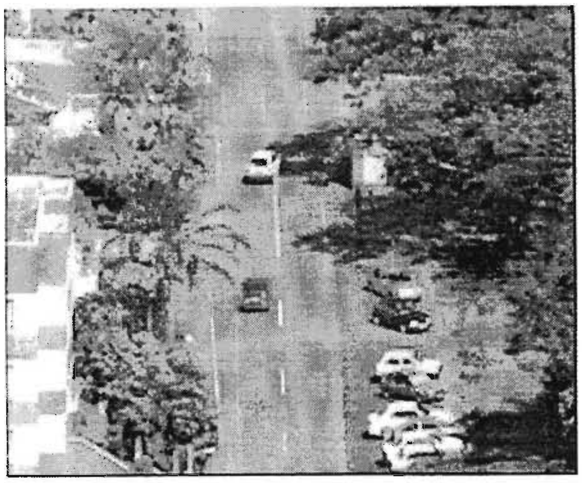
Figures 5-29 illustrates a selected set of decoded video frames that were recorded during the test. Figure 5-29(a) illustrates the decoded video frame from processing iteration eleven and Figure 5-29(b) illustrates the decoded result from iteration thirteen. Before the streaming video was paused, two objects were being tracked. However during the paused period A, illustrated in Figure 5-29(b), only one halted object was successfully tracked. The recorded % mean difference for the white vehicle on the left was recorded at 6.8% and the black vehicle on the right at 4.08%. During all testing, the threshold was set at 5% and as a result the white vehicle was effectively filtered. The cause of the white vehicle's 'high' percentage is attributed to the two pixel wide black stripe along its bottom edge. This stripe originates from the video frame border and was obviously included when the object was segmented. The significant contrast difference between the black stripe and the intensity of the surrounding road was obviously enough to increase the calculated mean difference beyond the defined threshold.



(a) (b)


(c)


(d)


(e)

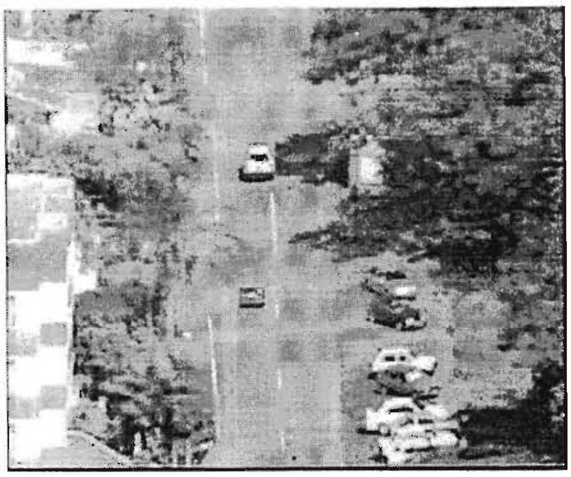

(f)

Figure 5-29: Resultant decoded video frames of the tracked halted objects in VS-2.

Figure 5-29(c) illustrates the resulting decoded video frame from processing iteration 27 while Figure 5-29(d) illustrates the decoded video frame resulting from iteration 28. The large increase in the amount of transmitted data in processing iteration 28 resulted due to the white vehicle being detected and segmented once again. Figure 5-29(e) illustrates the decoded video frame that resulted from processing iteration 37 while Figure 5-29(f) illustrates the decoded frame from iteration 38, the iteration before the pause button was pressed a second time. During this paused period, the white vehicle was tracked successfully but due to the black vehicle's partial occlusion by the overhanging tree, the tracking algorithm mistakenly assumed that it had decreased in size hence only a portion of the vehicle was displayed in the decoded frame. The increase in the amount of transmitted data to the decoder at processing iteration number 50 results due to the tracking algorithm updating the object data of the black vehicle.

As the white vehicle moved out of the video scene, the % mean difference was measured at 19.33%. Similarly, the % mean difference of the black vehicle, as it moved out of the scene, was measured at 11.27%. As both values exceeded the specified threshold, the correct decision was made by the algorithm in terms of deleting them from memory.

Figure 5-30 illustrates the resultant amounts of data that were transmitted per processing iteration of test video sequence VS-3. Once again, the streaming input video was paused in two places, indicated in

Figure 5-30 by A and B respectively. The % mean difference of the tracked halted object during the paused period A was measured at 3.99% and 0.18% during the paused period B.

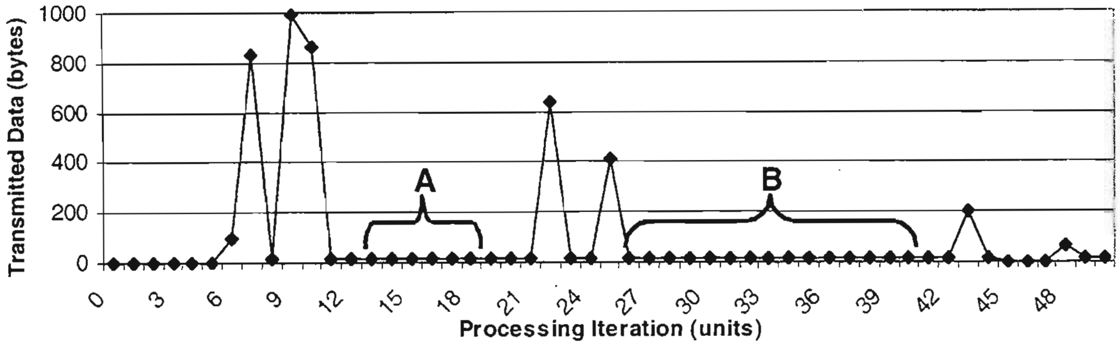


Figure 5-30: Resultant transmitted data of VS-3 when the streaming input video was paused.

Figure 5-31(a) illustrates the decoded video frame of processing iteration twelve, the iteration before the video was first paused. On this occasion, the algorithm successfully tracked the halted object throughout the paused period. An example of the tracked object is illustrated in Figure 5-32(b) which resulted from iteration sixteen. Figure 5-31(c) illustrates the decoded video frame of iteration 23. Although this iteration does not fall within either paused periods, it was included here in order to re-iterate the algorithm's ability to correct itself if it mistakenly tracks multiple instances of the same object. Following the execution of the search algorithm, the % mean difference of the old instance of the buffered object was measured at 15.99% and as a result was deleted from memory as it exceeded the 5% threshold. Figure 5-31(d) illustrates the decoded result of processing iteration 24. In the illustration it is noted that the old instance of the object was no longer tracked.

(a)

(b)

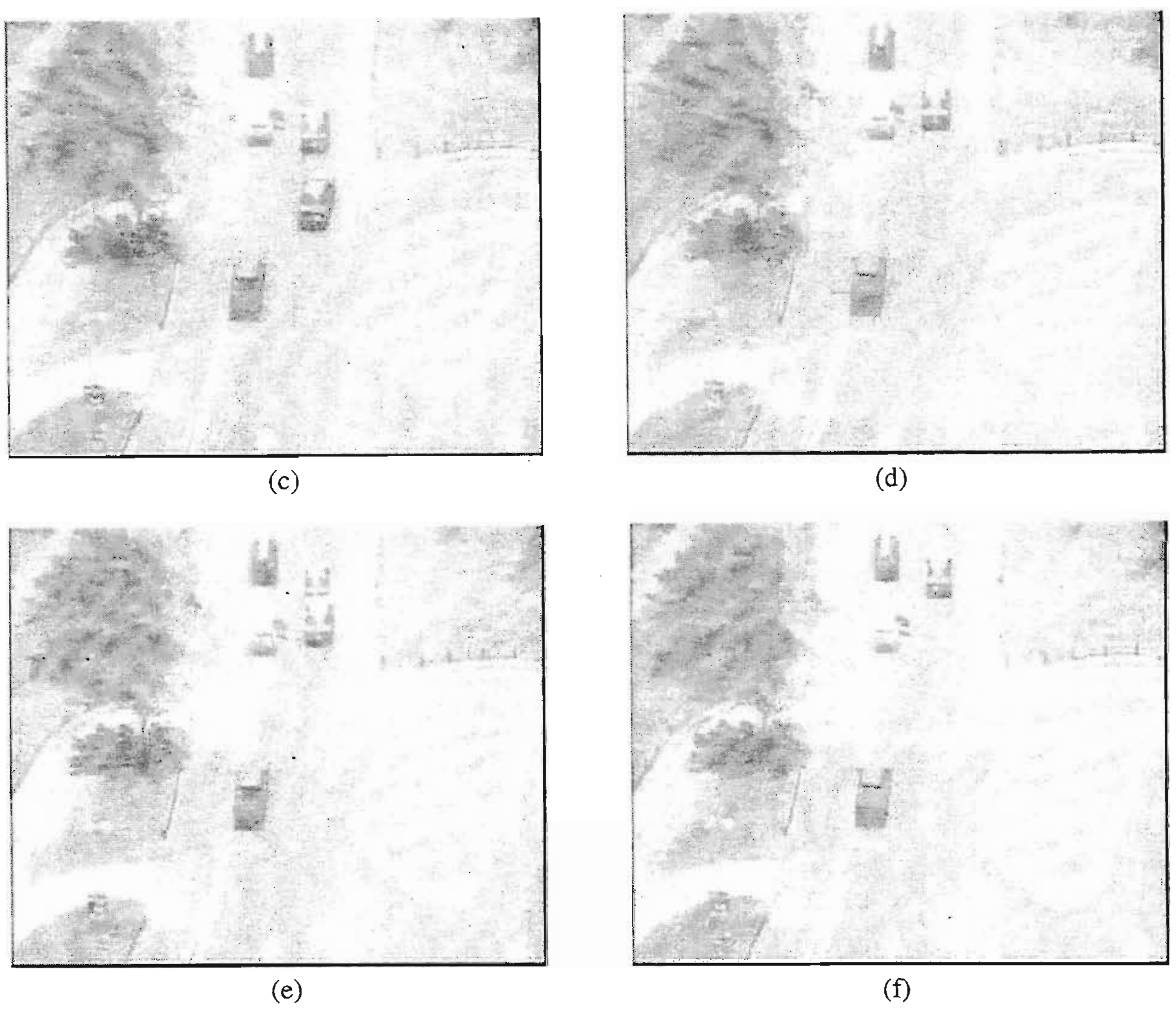


Figure 5-31: Resultant decoded video frames of a tracked halted object in VS-3.

Figure 5-31(e) illustrates the decoded video frame resulting from processing iteration 26, the first iteration of the second paused period. After measuring the % mean difference of the old instance of the tracked object to be 9.26%, the algorithm deleted it from memory and continued to track the current instance during the paused period. Figure 5-31(f) illustrates the decoded frame that resulted from processing iterations 27 to 40.

The different sets of results clearly illustrate the algorithms ability to track halted objects. Only on one occasion did the algorithm fail to track a halted object. This occurred in VS-2 when tracking the white vehicle on the left side of the road. However the reason for this mistake was due to a two pixel wide black stripe at the bottom of the segmented object that originated from the video frame's border. This stripe detracted from the accuracy of the executed search and as a result the halted object was deleted from memory. Otherwise the algorithm was able to successfully track all other instances of tracked objects hence satisfying one of the designs primary objectives.

5.3.5 Performance Compared to H.263

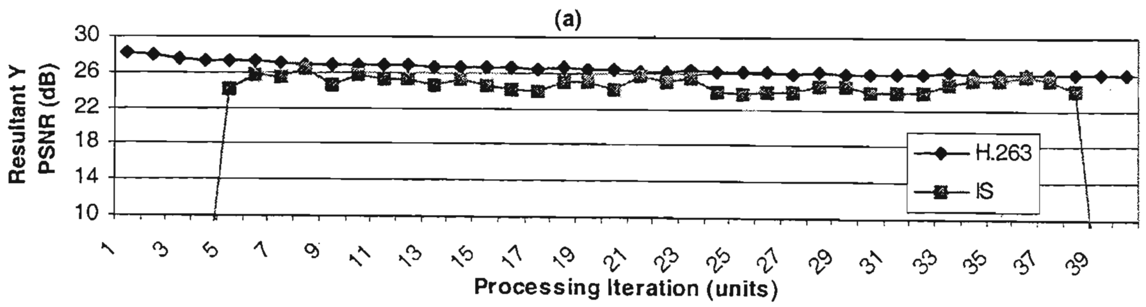
The performance of the implemented low bit-rate video compression system was tested against that of the H.263 low bit-rate video compression standard. The source code of the H.263 encoder and decoder was obtained from a research company called Telenor [Tele00]. During testing, the H.263 software was executed in the MSVC++ SDE.

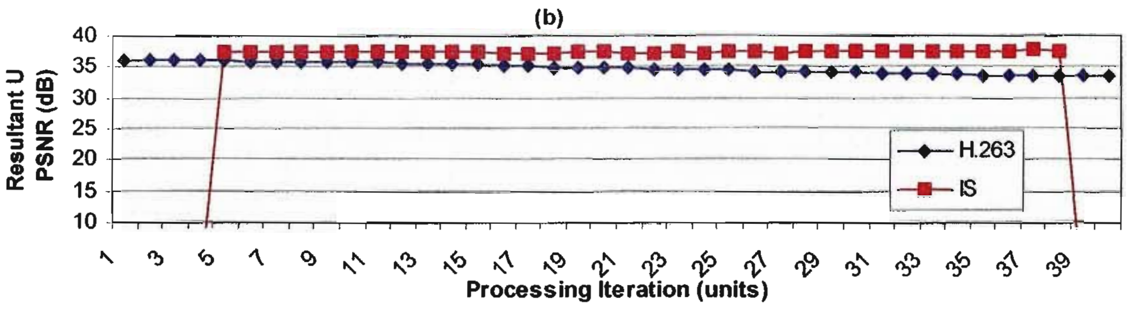
Throughout all of the previous evaluations, the executed software saved every YUV target video frame on the PC's HDD. One of the primary reasons for doing so was to ensure that the H.263 codec operated on exactly the same frames as the implemented system. In this manner, a comparison per processing iteration could be drawn between the implemented compression system and the H.263 standard. The H.263 software allowed the system operator to define certain compression parameters. Some of these parameters included:

- The ability to pre-define the quantisation parameter (QP) during encoding. As a result, the output bit-rate varies and the quality of the decoded video scene should ideally be consistent.
- The ability to set the encoder in 'auto' mode. In this mode, the quality of the decoded video is generally at its best at the expense of the variable, and usually high output bit-rate.
- The ability to pre-define the resultant bit-rate. In this mode, the encoder automatically varies the QP in order to maintain the defined bit-rate. The quality of the decoded video depends on the value of the bit-rate. The higher the bit-rate, the better the quality.

As far as comparing the performance of the implemented system and the H.263 codec was concerned, it was decided to use the average output bit-rate of the systems as the measurement benchmark. This was achieved by configuring the H.263 encoder so that the output bit-rate was approximately equal to that of the implemented system. In this manner, the rate metrics of the two systems could be compared based on the assumption that the compression settings of the two independent systems were similar. Therefore the evaluation procedure comprised the calculation of the average output bit-rate of the test sequences of the implemented system, and using these values to define the output bit-rate of the H.263 encoder. The bit-rate calculation did not include the affects of the reference frame. The reason for this was that even though a bit-rate limiter was implemented with regard to the transmission of the encoded frame data to the decoder, it was not designed with a feedback loop that automatically adjusted the QP as a function of the defined bit-rate. This however was the case in the H.263 encoder. The performances of the two systems were therefore compared by analysing the resultant PSNR per processing iteration as well as human impression.

Within all of the tests that were conducted, all of the H.263 advanced encoding options were enabled except PB-frame encoding. The reason for this was that when this option was enabled, a run-time error resulted. Several attempts were made to debug the software but the cause of the fault could not be located hence it was decided to disable this feature.





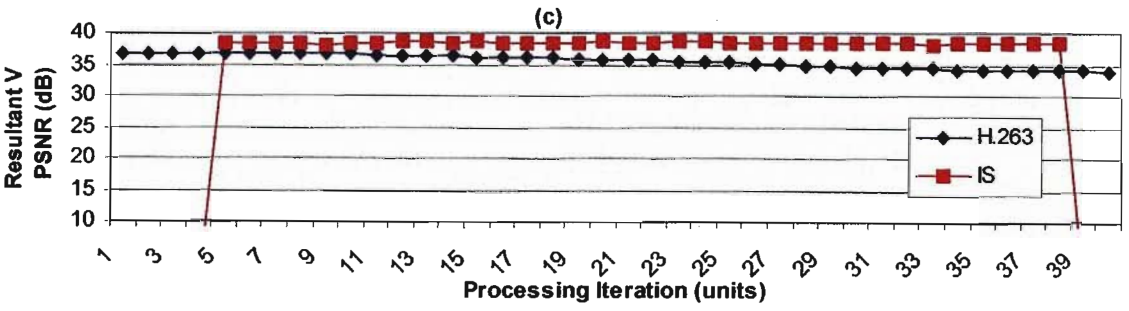
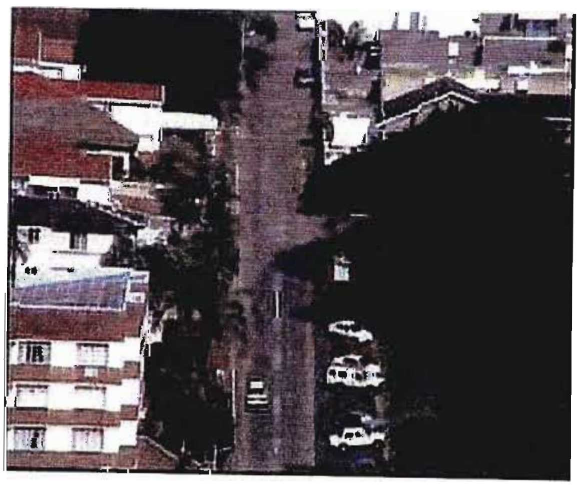
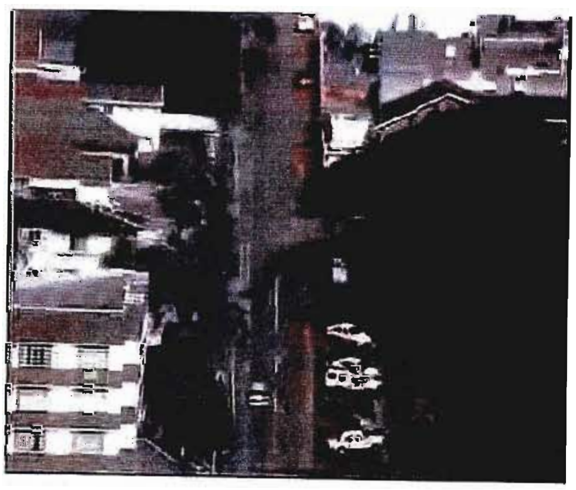


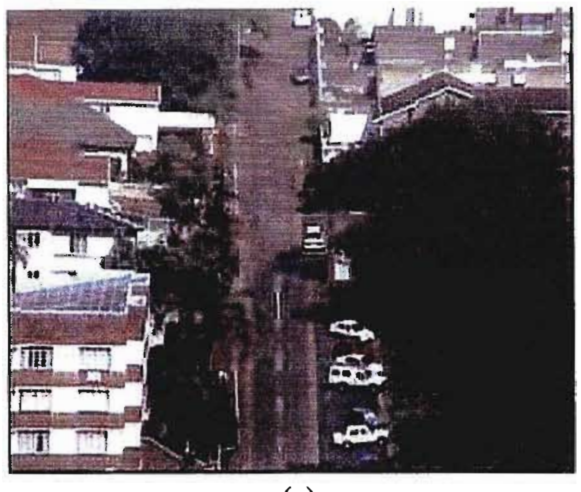
Figure 5-32: PSNR comparison of the implemented system to H.263 of VS-1.

Figure 5-32 compares the PSNR results that were obtained from applying test video sequence VS-1 to the implemented video compression system and the H.263 compression standard. Note that 'IS' within the legend denotes the Implemented System. The H.263 bit-rate was set at 760 bits/second as this was the calculated average output bit-rate of the implemented system.

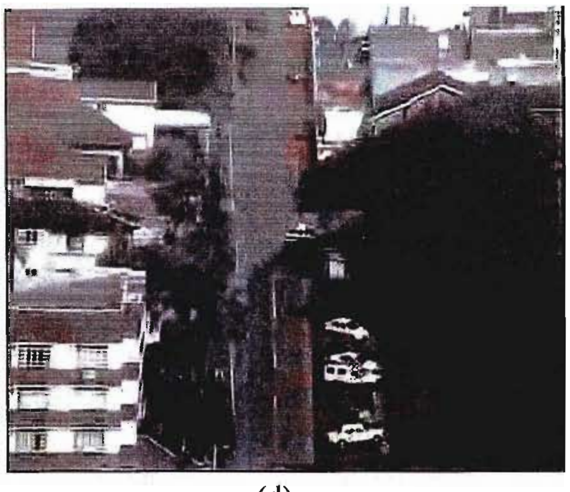
From the results depicted in Figure 5-32, it would appear as if the performance of the systems are reasonably similar. However, the performance is possibly best evaluated by comparing examples of resultant decoded video frames. Figure 5-33 illustrates the decoded video frames from two randomly selected processing iterations. The frames on the left are those of the implemented system while those on the right are their H.263 counterparts. The frames resulted from iterations eighteen and 34 respectively. From the illustrations, the superiority in terms of visual quality and colour of the frames obtained from the implemented system is clearly evident.


(a)


(b)



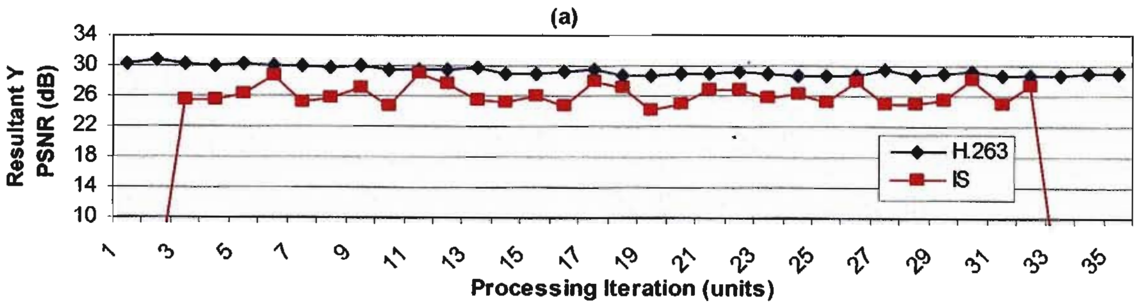
(c)

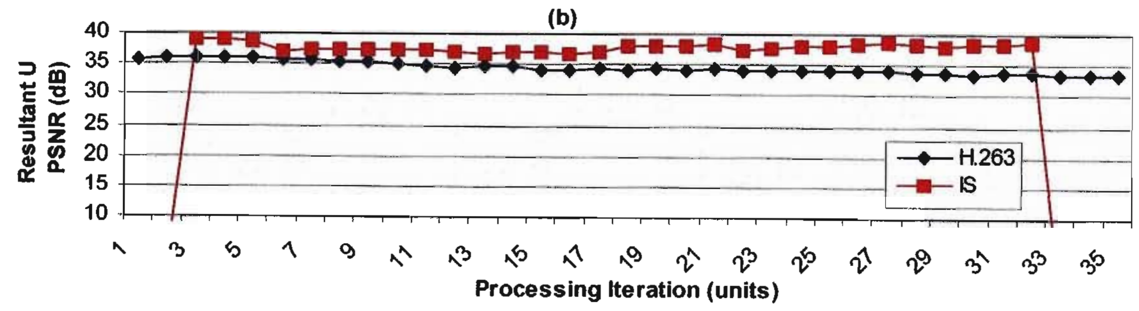


(d)

Figure 5-33: Comparison of VS-1 decoded video frames of the implemented system and H.263.

VS-2 was tested in a similar manner. In this case, the average bit-rate of the implemented system was calculated at 4,833 bits/second. The results are similar to the previous set in that the Y PSNR of the implemented system is once again consistently less than the H.263 standard, while the U and V PSNR are consistently greater.





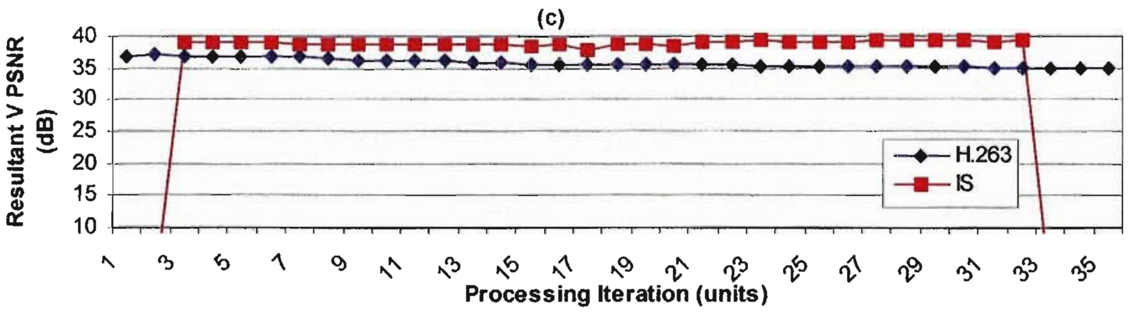


Figure 5-34: PSNR comparison of the implemented system to H.263 of VS-2.

On comparing examples of the decoded video frames, the difference of the two systems in terms of clarity and colour is again noticeably different. Figures 5-35(a) and (c) illustrate the decoded video frames from iterations seven and nineteen respectively that were obtained from the implemented system while Figures 5-35(b) and (d) illustrate their H.263 counterparts.

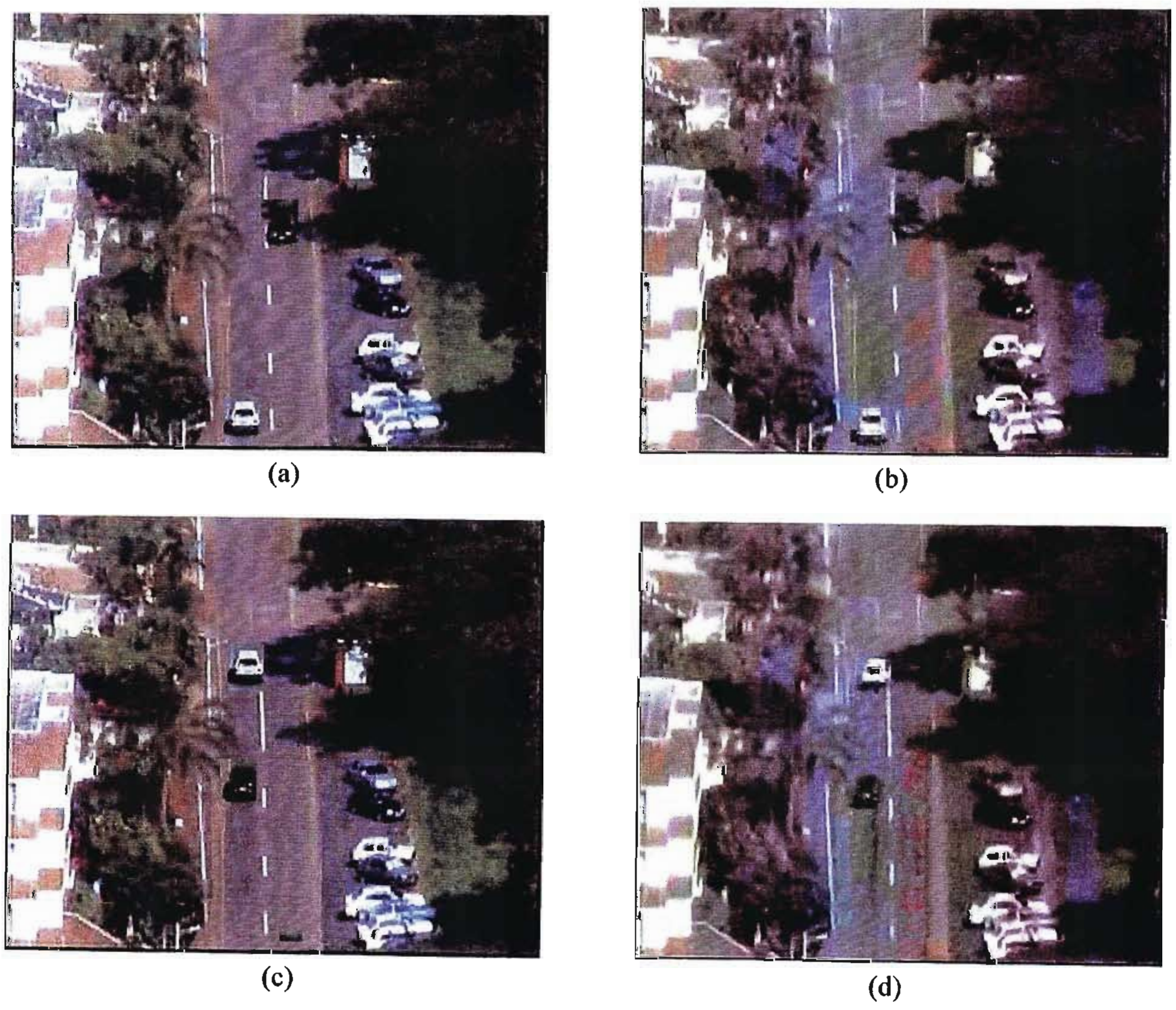


Figure 5-35: Comparison of VS-2 decoded video frames of the implemented system and H.263.

Figure 5-39 compares the results that were obtained from applying test video sequence VS-3 to the implemented video compression system and the H.263 standard. In this case, the output bit-rate of the

H.263 standard was set at 7,668 bits/second as this was the calculated average of the implemented system. Examples of the decoded frames of both systems are compared in Figure 5-37. Once again the colour degradation of the H.263 video frames is the most prominent difference between the two systems.

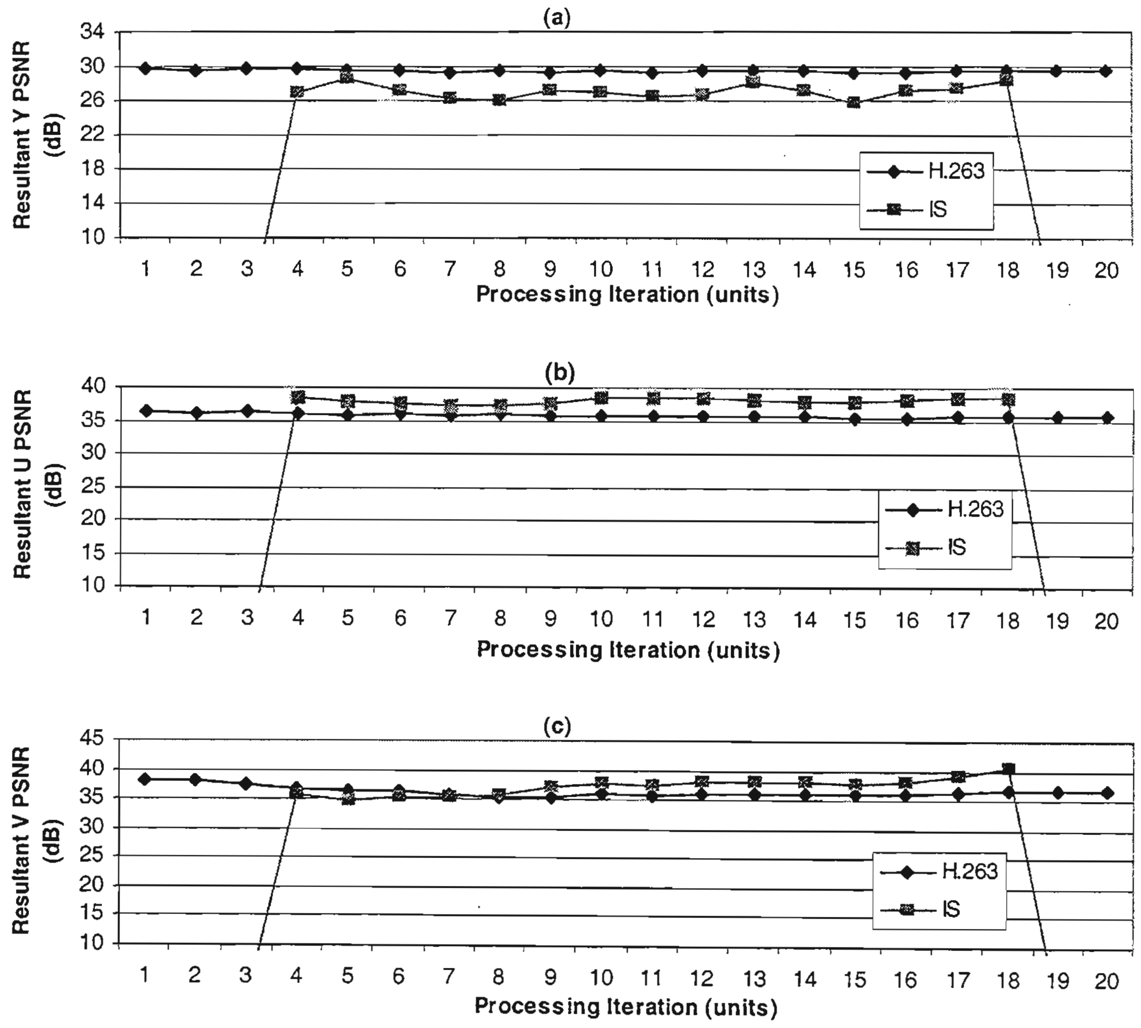


Figure 5-36: PSNR comparison of the implemented system to H.263 of VS-3.

On a frame per frame basis, it can be concluded that the implemented system outperforms the H.263 standard in terms of visual quality and colour retention in all three cases. However it must also be realised that the two systems were compared based on the assumption that their respective compression settings were similar.

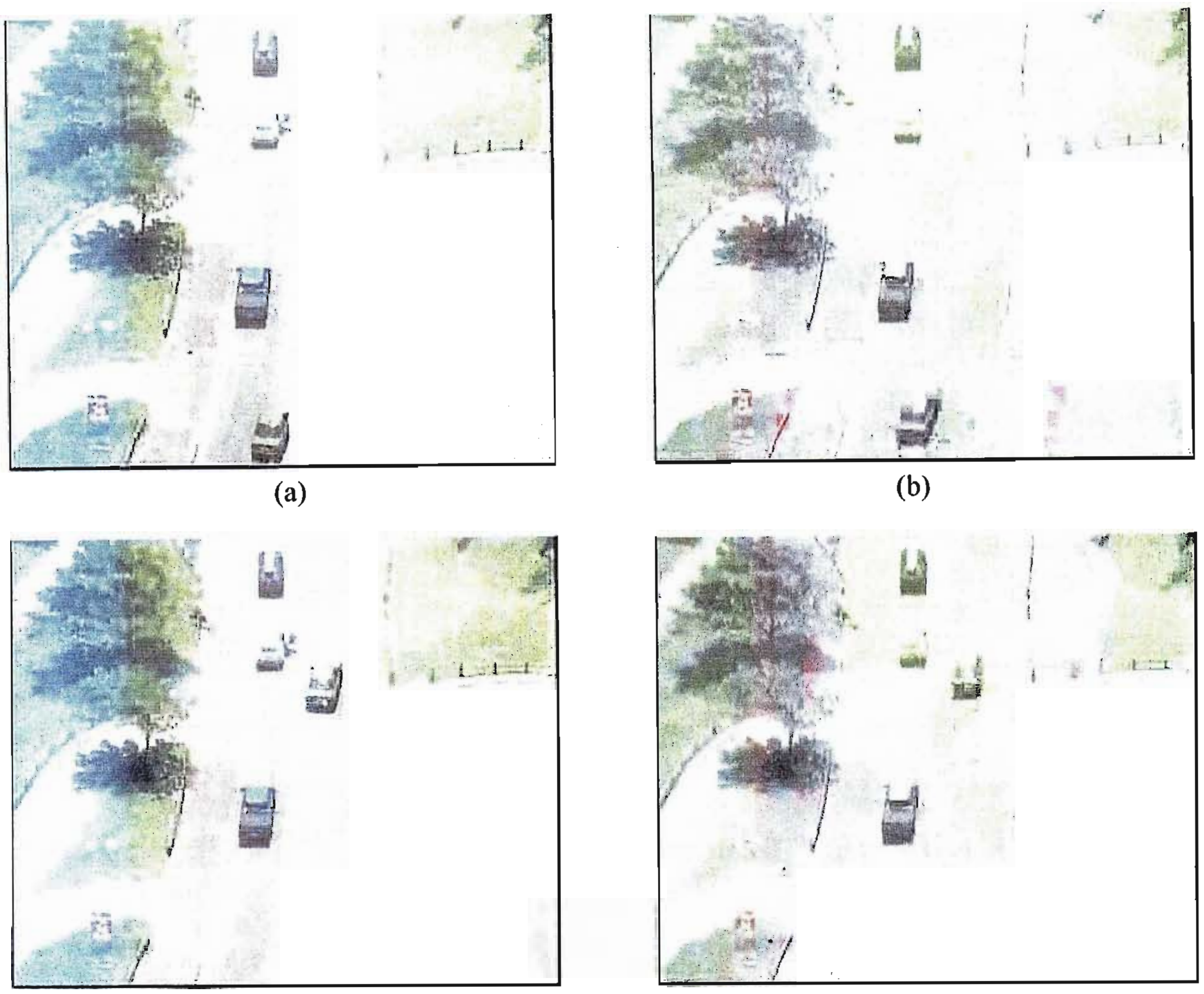

(a) (b) (c) (d)

Figure 5-37: Comparison of VS-3 decoded video frames of the implemented system and H.263.

In all three test cases, it would be expected that because the PSNR results of the different systems are reasonably similar, the quality of the decoded video frames would be comparable. However from the obtained results, this is clearly not the case. In order to explain the unexpected results, we need to analyse the PSNR graphs more closely. In each test, the Y PSNR values of the implemented system are either equal to or at most approximately 3dBs less than that of the H.263 system. On the other hand, the U and V PSNR values of the implemented system are in most cases at least 3dBs greater than the H.263 system. Now on analysing the decoded video frames, it is clearly the colour components of the H.263 system that are distorted and not the spatial detail that is contained within the Y component. It is this colour distortion that renders the overall image as aesthetically displeasing. Therefore it can be concluded that the implemented system handles the compression of the colour U & V components better the H.263 standard.

This comparison also highlights the potential inaccuracy of PSNR as a measurement index of perceived image quality as 3dBs does not necessarily represent a vast difference in terms of image quality. Hence the reason as far as image and video processing is concerned, results should be both objectively and subjectively quantified.

5.4 Bandwidth Allocation

This chapter has provided the results for the two main components of the implemented system, namely the reference frame encoder/decoder, and the object tracking system. Now that we have a feel for the bandwidth requirements of the two components, we can analyse how the available bandwidth of a real-life system could be divided up between the two components. By doing so, we could automate the updating of the reference frame and ultimately we would be able to predict its frequency.

$$update\_time = \frac{\left(\frac{152{,}064}{g}\right) * 8}{h} \qquad (5\text{-}5)$$

The time between automated reference frame updates over a bandwidth limited channel can be calculated from (5-5) where g = the desired compression ratio of the reference frame and h = the bandwidth of the channel that is reserved for the transmission of the frame from the encoder to the decoder. As an example, consider a bandwidth limited channel of 16 kbits/second. Looking at average bit-rate requirements, the third object tracking test that was conducted, i.e. VS-3, was the worst case situation in that it had the highest average bit-rate requirement of 7,668 bits/second. If we had to round this value up and reserve 8 kbits/second for the transmission of object data, then we would have a further 8 kbits/second for the transmission of the updated reference frame (h in (5-5)). Furthermore, if we required the reference frame to have a compression ratio of 25 (g), then from (5-5), the reference frame would be automatically updated every 6.08 seconds.

Obviously there are two main factors that determine the update frequency. Firstly, the lower the compression ratio of the reference frame, i.e. the bigger it's size in terms of bits, the longer the time required to transmit it hence it is updated less frequently. Conversely, the higher the compression ratio, the more often it would be updated. Secondly, the greater the portion of the available bandwidth that is reserved for the reference frame's transmission, the more frequent it would be updated and vice versa. Therefore in terms of a complete system, these two factors need to be carefully considered in order to ensure optimal usage of the total available channel bandwidth.

5.5 Summary

This chapter provides the results of the implemented low bit-rate video compression system. It begins by describing the measurement criteria that were used to evaluate the performance of the system both objectively and subjectively. The objective measurement methods included the calculated bits/pixel, compression ratio, bit-rate and PSNR values of the decoded video, while the subjective method was that of human impression. Although most results are traditionally based on mathematical calculation, such measurements alone cannot be used to consistently and accurately categorise decoded images and video. This limitation of doing so became more apparent as the chapter progressed. Therefore a holistic approach was adopted in that the results were judged according to both mathematical calculation as well as aesthetic appeal.

The chapter was divided into two main sections. The first section presented the results that were obtained from the evaluation of the implemented reference frame encoder and decoder while the second section presented the results of the object segmentation and tracking algorithms. Three different test frames were used to evaluate the performance of the frame encoder and decoder. The initial evaluation tests involved the variation of each of the four user-defined compression variables one at a time. Thereafter they were varied simultaneously and the combined results were discussed. The goal of the tests was to obtain a set of compression variables that would optimise the resultant bits/pixel value of an encoded frame while at the same time, retain a reasonable level of decoded fidelity. Rather than specifying exact values, a range of compression values was suggested as it was determined that the performance of the encoder varied according to the contents of each frame. A discussion of the measured encoding and decoding times followed. Finally, a comparison between the performance of the implemented system and the JPEG still image compression standard was presented. The results were presented in a graphical form of measured PSNR as a function of bits/pixel. It was concluded that at reasonably high bits/pixel values, the JPEG standard outperformed the implemented system however at bits/pixel values ≤ 0.28, the opposite was the case.

The second section presented the results of the implemented object segmentation and tracking algorithms. It began by evaluating the effects of varying the two user-defined segmentation variables. The results indicated that the variables were not very effective in filtering ambient noise within the video scene. A more effective means of filtering ambient noise was the concentration of the segmentation algorithm within a user-defined area of the video scene. By doing so, noise that fell outside of the defined area that could have been mistaken for objects in motion was ignored thus averting erroneous object segmentation.

Three different test video sequences were used to evaluate the performance of the object tracking algorithm. The algorithm's performance was measured according to the resulting bit-rate, amount of data transmitted, YUV PSNR and the processing time. From the examples of the decoded frames that were presented, the results indicated that the tracking algorithm performed best in situations where the objects were reasonably small and moved at slow to moderate speeds. The results indicated that the tracking algorithm was also able to tolerate rotational motion, even though it was not specifically designed to do so. The second video sequence illustrated the algorithm's ability to track multiple objects at the same time. A weakness that was identified in this test sequence was the algorithm's inability to accurately track objects that fell within shadows. The reason for this being the low contrasts between the objects in the shadows and the colour of the road.

The tradeoff off of the algorithm's performance in terms of an object's size and speed at which it moves was illustrated within the last test video sequence. Finally the chapter ended with a comparison between the implemented video compression system and the H.263 video compression standard. As the architectures of the two systems are different, it was decided that the fairest way to compare the results of the two systems was to equate their average output bit-rates hence the systems would operate under similar circumstances. The systems were also set-up so that they each operated upon exactly the same input video frames. From a measurement point of view, the resulting YUV PSNR values were compared and in all three test cases, the ratios were similar. However from a human impression point of view, the colour and visual quality of the decoded video frames from the implemented system were superior to that of the H.263 standard.

Based on the observed characteristics of the implemented system, the following chapter presents several recommendations that could possibly improve its performance.

CHAPTER 6

FUTURE RECOMMENDATIONS

As is so often in the case of development work, looking back in hindsight armed with recently acquired knowledge, you cannot help but ponder past decisions made. The only extraordinary thing about questions such as *'What if I had done this'*, or *'What about this idea'* along with many others, would be if they weren't asked in the first place. This chapter answers similar questions by recommending possible improvements to the implemented system as far as performance and user friendliness is concerned. Furthermore, new avenues of research along similar lines as those investigated in this work are identified.

Code optimisation is normally always an issue as far as software systems are concerned and in this case there is no exception. Considering execution times, the implemented segmentation and tracking algorithm is not really of concern in this regard as each processing iteration required on average less than 40 ms. Obviously this value does not take into account the 160 ms required to capture the five consecutive input frames. However a more pertinent issue is the amount of time required to encode and decode a frame. The results showed that depending on the user defined variables, to encode a frame required anywhere from 105 to 250 ms while decoding required between 70 and 90 ms. Clearly there is room for improvement here.

As mentioned in Chapter 5, the 2D DCT is the most computationally expensive procedure involved during the encoding stage. Another computationally expensive procedure is quantisation as it involves a number of divides. Optimising these processes would therefore go a long way as far as overall execution times are concerned. An interesting technique that was located subsequent to the system's implementation is that proposed by [Docef02]. In this paper, Docef *et al.* proposes the combination of the DCT and quantisation stages. Furthermore, they also propose a multiplierless integer-based DCT implementation. This is achieved by using combinations of adds and shifts that would normally be executed faster than multiplies on most processors. The results that were presented were promising and in some cases, execution times were reduced by up to 50%. This is just one of many techniques that would undoubtedly reduce encoding times as well as impact on decoding times.

Another avenue that deems further investigation is perceptual distortion. The current system requires a user to define four variables that determine how a frame is encoded. From the obtained results, the conclusion was drawn that no one set of values could be specified that would optimise the compression ratio for all images. This is due to the content dependent nature of the JPEG compression standard on which the implemented system is based. Therefore a possible improvement would be an encoder that adapts to the scene at hand according to a defined distortion level. A technique along a similar line that may be worth investigating is that proposed by [Hontsch02]. In this paper, DCT quantisation factors are dynamically altered as a function of locally computed image frequency, orientation and spatial characteristics. The successful implementation of such a system only requires one variable to be defined by a user as opposed to the current four. This would obviously aid user friendliness.

A further improvement that can be made to the frame encoder would be to automate the updating process. Currently a reference frame is updated manually and in the long run, this could become quite tedious. A means of automating the process would be to enable a user to define a refresh period so that at the end of each period, a new frame would be captured and would serve as an updated reference. However considering the current design of the system is basically an evaluation platform, in order to make the transmission of the updated reference frame to the decoder as realistic as possible, a feedback bit-rate controller coupled with an output buffer should also be used. The purpose of these functions would be to simulate a bandwidth limited transmission channel. The bit-rate controller would monitor the data channel and during transmission lulls, packets of the updated frame would be transmitted. In this manner, bandwidth utilisation would be optimised.

Results indicated that the most effective form of ensuring that ambient noise within a video scene was not erroneously segmented was to confine the area within a video scene in which the segmentation and tracking algorithm was executed. At present, the restricted area is limited to the shape of a rectangular block. A more realistic approach would be to allow the system operator to define the area as a polygon. The advantage of doing so would enable any arbitrarily shaped travelled path to be selected.

There is always room for improvement as far as object segmentation is concerned. Unfortunately, the efficiency of proposed algorithms can normally only be validated by experimentation. The reason for this is that segmentation algorithms tend to be application specific and when applied in slightly different situations, their operation is not guaranteed. There are a multitude of different algorithms to choose from. These range from segmentation by means of optical flow [Marsh94], edge detection [Canny86] and [Smith97] to morphology [Weeks96] and clustering [Krav99]. Clustering was implemented within this work because of its relative simplicity and fast execution. However a shortcoming of this method was its tendency to under segment objects. Subsequently a morphological filter had to be implemented in order to refine each segmented object. Although effective, this method is not optimal. An improved system would obviously be one where an object is segmented accurately first time around. A possible method to achieve this would be to use a combination of colour and luminance information and not only luminance as in the case of the implemented system. Recall that preliminary testing indicated that little to no motion was conveyed within *differenced* chrominance frames, however this does not mean that the chrominance components cannot be used during the actual segmentation stage. A similar technique that was found after the current system's implementation was proposed by [Ohm97]. In this paper, motion and colour information was used to segment objects. The results that were presented appeared promising. This is a potentially improved segmentation concept but once again, only experimentation would prove its suitability to the intended applications of the current system.

A further enhancement would be to enable the system to cater for objects undergoing rotational motion thereby improving the accuracy of a decoded sequence. The limitation of the current segmentation algorithm is that it segments objects within a rectangular block. An improved system would allow objects of any shape to be segmented. As an example, consider a similar situation to test video sequence one (VS-1) where an object travelled up a road and then around a corner. Instead of superimposing the outdated view of the object onto the reference frame as the object moves around the corner as illustrated in Figure 6-1(a), an improved algorithm would track and update the object as it changes shape due to it undergoing rotational motion. An example of this process is illustrated in Figure 6-1(b).

A means of improving the current object tracking technique would be to introduce a form of motion estimation similar to that employed within MPEG-1/2 [MPEG-01] or H.263 [H.263]. If this were the case, instead of tracking a complete object, as is currently the case, the individual 8x8 blocks that make up the object would be tracked. By doing so, motion vectors corresponding to each block coupled with their predicted error component would be transmitted to the decoder. The output bit-rate of this method should be less than the current method, as complete objects would no longer have to be transmitted every time they undergo a change. The disadvantage of this method would be its associated increase in computational effort but this could be a suitable tradeoff.

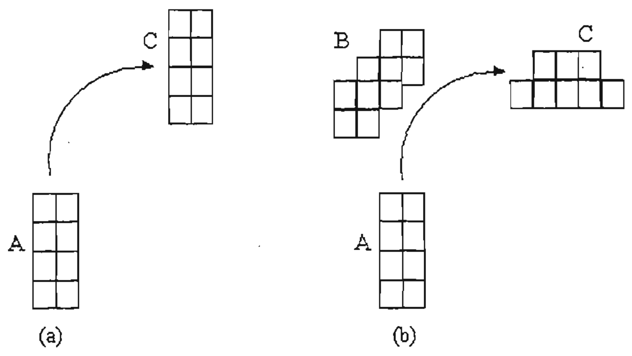


Figure 6-1: Ability to cater for rotational motion, (a) current tracking process and (b) recommended tracking process.

A further advantage of tracking the individual blocks of an object is that it would be able to handle occlusion better than the current system. Instead of retransmitting the complete updated object to the decoder each time it had to change shape, if the tracked object had to become partially occluded for example, then only those blocks that are affected would need to be updated at the decoder. At the same time the remaining macroblocks would remain unchanged. The process would also have a positive impact on output bit-rate.

Finally, one of the characteristics of the tracking algorithm is that a tradeoff exits between the size of an object and the maximum speed that it can travel at in order for it to be successfully tracked. Currently the size of the motion tracking area that governs this tradeoff is hard-coded in software. A better option would be to enable a user to define the approximate size of the objects as well their estimated maximum speeds. Based on these parameters the system would then be able to calculate the correct motion tracking area to be used. This would guarantee the successful tracking of objects in any situation.

CHAPTER 7

CONCLUSION

At the outset of this work, it became apparent that the presented research topic of low bit-rate video compression was extremely broad hence some refining would be necessary. The application and techniques that would ultimately be implemented as a result of the investigation were also left unspecified as it was deemed that these would evolve during the literature review. It was eventually decided that the system to be implemented would be application specific in that it would focus on the concept of vehicle tracking. The system would include the ability to compress video on a per frame basis as well as being able to identify and track vehicles within a mostly static scene. All video scenes would be filmed by a stationary video camera. A typical example of the system's intended application is that of a suburban road on which travelling vehicles exhibit mostly translational movement. The focus of the project therefore, was to determine the compression techniques that would be best suited to this application and as a result, achieve the low bit-rate output goal. The final implementation of the system can effectively be divided into two different sections. The first section addresses spatial compression, in other words the compression of an individual video frame and the second section deals with the segmentation and tracking of vehicles within a video scene. Although the two sections are distinctly different, they have been married together as the implemented system uses a compressed video frame as a static background scene on top of which objects are tracked.

The motivation for this topic of research is that the benefit of being able to separate a video scene into differing areas of interest, i.e. a background scene and objects that move with respect to it, is becoming more apparent in today's digital age. Such benefits include very low bit-rates, improved retention of decoded image fidelity as well as the ability to interact with the video itself. One of the main driving forces behind the continued effort to improve the efficiency of compression algorithms is the growing demand of multimedia services. Considering the important role that digital video plays within this field, a major problem is the huge demand that it places on storage and bandwidth resources. As a result, there is a trend nowadays to steer away from the traditional way of thinking and to divide video into separate entities or objects that can be individually manipulated. This allows for very low output bit-rates as well as making the transmission of video over low bandwidth channels more practical.

This dissertation began by discussing the need for video compression with a bias towards application specific compression systems. Thereafter an insight into the composition of digital video and the philosophy behind its manipulation with regard to the human visual system's limitation to high frequency changes in colour was given. The technique of colour sub-sampling was also described, as it is one of the most commonly used methods to discard redundant image and video information. In this manner, there is less data to encode and therefore has a favourable impact on output bit-rate.

Having explained the fundamentals of video in detail, the theory behind the concept of compression was then outlined. After describing the two general categories in which compression is divided into, namely *lossless* and *lossy*, an overview of some of the more common approaches to image compression was then presented. The need for explanation of the two compression categories is that many video compression algorithms including existing standards are strongly based on techniques used in the compression of still images. It was therefore deemed prudent to familiarise the reader with the compression fundamentals before moving onto a discussion of the techniques used to compress video. The discussion introduced the concept of transforms and outlined the characteristics of the more common Discrete Cosine Transform [Ahmed74], Fractals as well as Wavelets [Polikar99]. An overview of the JPEG [JPEG01] still image compression standard was presented, as the video frame compression system that was ultimately implemented was based upon those techniques used within the standard. One of the major differences however was the concept of a variable block into which a video frame would be sub-divided before being compressed. This technique as well as its considerations was described as it was ultimately to be incorporated into the system.

Although no temporal compression was implemented, it was deemed that a discussion on video compression would be incomplete without at least providing an overview of some of the more commonly implemented techniques as well as the considerations that are involved. The discussion centred on the concept of inter-frame encoding, but more specifically motion estimation. This section was concluded with an overview of existing video compression standards such as MPEG-1/2/4 [MPEG-01], [MPEG-4-01] and H.263 as all of these standards implement some form of motion estimation. Another reason for describing the H.263 [H.263] standard is that it would eventually be used as a comparison against the implemented system.

The concept and considerations of objects was then elaborated upon. The discussion pointed out that object segmentation is a non-trivial task and that the efficiency of most algorithms tends to be scene dependent. Several techniques were described including Canny edge detection [Canny86], the SUSAN method [Smith97] as well as seed growing [Adams94]. Another technique that is also used is that of morphology. Subsequent to its segmentation ability, morphology can also be used as a noise filter. A brief overview in this regard was presented, as the filtering technique of morphological *opening* was later to be incorporated into the implemented system. The discussion then moved onto cluster segmentation. Clustering is a process of grouping similar entities, pixels in the case of images, which belong to a common object. Four clustering techniques were introduced but the discussion focussed mainly on the technique of *density* clustering [Krav99]. There are two distinct advantages to using density clustering. Firstly density clustering converts an input image to what was termed a *clustermap* on which the algorithm then operates. The advantage of doing so is that the process is quick in execution as the clustermap is a greyscale image that is 64 times smaller than the input image. The second advantage is that during the construction of the clustermap, neighbouring pixels are averaged and this has a positive effect of filtering random noise. Based upon its simplicity and the advantages that it offers, object segmentation by means of clustering was considered the method of choice. The review concluded with a discussion on object tracking involving several different techniques ranging from optical flow to Kalman filtering. It was ultimately concluded that the concept of tracking is non-trivial and that most algorithms are designed with a specific use in mind.

Having laid a solid theoretical foundation, the next step was the design of the implemented system. Before delving into the mechanics of the system, its goals, general specifications as well as the assumptions that were made were outlined in order for the reader to have a clear understanding of the design choices made. Some of the key requirements included a low bit-rate output, the ability to operate on streaming input video in CIF resolution as well as the decoded output being in true colour. Important assumptions that were made included the camera remaining in a stationary position at all times and that the transmission channel used would be error free.

Having formalised the specifications, a high level overview of the implementation was then presented. It was described how the system would separate streaming input video into two categories that included a static reference frame (background scene) and objects that move in relation to the reference. As a consequence, the function of the encoder was twofold. It would initially encode a reference frame and transmit this to the decoder where it would be decoded and buffered. Thereafter throughout the following processing iterations, the encoder would segment objects within the video scene and track them in relation to their previous segmented positions. Their calculated positional offsets, represented by motion vectors, are thereafter transmitted to the decoder. The decoder would then superimpose the buffered version of each object according to its updated motion vector onto the reference frame and display it to the system operator. The continual execution of this process would result in an almost life-like decoded video sequence being viewed by the system operator.

Following the definition of the system, its implementation could commence. The section began by providing an overview of the utilised hardware. The core component in the system is the PCI TriMedia DSP video capture and display card. An important attribute of the TriMedia is its ability to share data between itself and its host PC across the PCI bus. Due to its ADC front end, it was therefore implemented as the encoder while the host PC was implemented as the decoder. Subsequent to being the decoder, the host PC was also used as a GUI. Its purpose was to display the decoded video as well as enabling the system operator to configure compression variables. Calculated rate metrics including PSNR, bit-rates and others were also displayed on the GUI.

An in-depth discussion on the software implementation followed. Where possible, the implemented techniques were adapted in order optimise processing efficiency. As an example, instead of calculating the 1D DCT of every row and then column of a block to be transformed into the frequency domain following the row-column approach, the adapted algorithm only calculated the required DCT's determined by the implemented coefficient limiting algorithm. A table illustrating the computational saving indicated that this concept was most favourable. The next implementation stage was that of the object segmentation and tracking software. Preliminary testing was used to verify the accuracy of the original proposed techniques. The preliminary investigations revealed that the original idea of detecting motion by differencing all three of the YUV components of consecutive input frames would not be adequate for two main reasons. Firstly, the differencing of consecutive frames did not convey enough motion information hence the segmentation algorithm would be unable to operate correctly and secondly, even after increasing the number of frames which were differenced over, the U and V components still conveyed little to no motion information. After performing a mathematical analysis, it was determined that in order to segment reasonably small objects travelling at a minimum speed of 20km/h, the Y component over five input frames would have to be differenced. The final stage of the encoder implementation was the object tracking algorithm. The algorithm was reasonably complex, as it had to be able to manage existing tracked objects, encode newly segmented objects as well as being able to handle objects that changed in size as they entered or exited a video scene.

As a consequence of its design, the tracking algorithm exudes a tradeoff between the size of an object and the maximum speed at which it can travel in order for it to be successfully tracked. An analysis was carried out in order to determine the relationship of this tradeoff so that when the system's performance was evaluated later, its limitations were understood. It was found that the bigger the size of a tracked object, its maximum permissible speed at which it could travel for it to be successfully tracked was reduced. The opposite was the case if an object decreased in size.

Having implemented the design, a series of tests were conducted in order to evaluate its performance. It is most often the case that image and video systems are quantified by mathematical measurement alone. However research ([Teo94], [Osberg97] and [Damer00]) has indicated that these traditional measurements are not necessarily the most accurate. Take the calculation of PSNR for example, it is basically the measurement of the difference between two pixels, i.e. corresponding pixels of a decoded frame and it's original. Humans however don't conceptualise a decoded image or video sequence on a per pixel basis, they view it as whole entity. Therefore from a holistic point of view, the decoded video was classified according to both mathematical measurement as well as what was termed human impression.

The testing procedure was broken into two main categories. These included the testing of the video frame encoder and decoder followed by the object segmentation and tracking algorithms. Three different colour video frames comprising different levels of spatial density were used during the testing of the frame encoder and decoder. The conducted procedure initially varied each of the four user defined variables independently of each other. The purpose of this was to determine the effect that each had on the decoded frame. Thereafter they were varied simultaneously. The goal of the tests was to obtain a set of variables that under all circumstances maximise the compression ratio whilst still being able to retain reasonable levels of image fidelity. It was however found that no one set of parameters when applied to the different images satisfied this goal. This was not unexpected considering the algorithm is based on the JPEG standard whose encoding efficiency tends to be content dependent [Dins90] [Vais92]. Therefore as a compromise, a range of variables were recommended and it would ultimately be up to the user to configure the system to his or her liking.

Following a discussion and analysis of the encoder's dependence on the DCT in terms of encoding times, the performance of the implemented system was compared to the JPEG standard. Only the Y components were compared and it was concluded that at moderate to high bpp values, the JPEG standard's PSNR performance was only slightly better than the implemented system. The PSNR performance of the system was at worst 3 dBs less than JPEG. The reason for this was that at these bpp values, more coefficients per block would remain after encoding and considering the JPEG blocks are small in size, the retained quality per block was greater than the larger sized blocks that were utilised by the implemented system. On the other hand, the performance of the system exceeded the JPEG standard at bpp values < 0.29.

The second part of the testing was concerned with the system's ability to segment and track vehicles within a video scene. Three test sequences were used and in each case, the vehicles had different sizes and moved at different speeds. The system requires a user to define two threshold values whose purpose is to filter ambient noise within a video scene. The test results indicated that although they had some impact on the accuracy of segmented objects, the most effective filter however was that of the user defined boundary tracking area that confined the executed segmentation and tracking algorithms. In all three tested scenes, the vehicles of interest were successfully segmented. The only time the system failed to segment a vehicle correctly was when it fell in the shadow of an overhanging tree. However it must be mentioned that the situation was complicated further by the vehicle being black in colour as well as falling in a shadow. On scrutinising the video scene, it was noted that the vehicle was barely distinguishable from its surroundings thus the inaccuracy was acceptable.

The tracking performance of the system was then evaluated. The first test video sequence comprised a low activity scene in which a small vehicle slowly travelled up a road that was positioned in the centre of the scene. In this scenario, the system performed very well in that it successfully tracked the object as it entered the scene, travelled up the road and then exited. An average output bit-rate of 760 bits/second over the sixteen second duration was recorded. The system's inability to cater for rotational motion was demonstrated as the vehicle travelled around a corner at the top of the road. It was however recommended that an improved system would be able to cater for this form of motion.

The second test video sequence comprised larger vehicles travelling at higher speeds compared to the first sequence. One of the reasons for choosing this sequence was that it contained three moving vehicles and would therefore test the system's ability to track multiple objects simultaneously. On the whole the tracking results were satisfactory and an average resultant bit-rate was calculated at 4,833 bits/second over the fourteen second duration. However two important characteristics were noted. Firstly, the affects of partial occlusion were witnessed as the algorithm 'chopped' a part of a tracked vehicle off when it moved behind some overhanging branches in the road. Once again this property was purposely omitted from the design hence the result was anticipated. Secondly, the algorithm incorrectly tracked two instances of the same object as it moved within the shadow of an overhanging tree. Following the execution of the implemented search algorithm, it eventually corrected itself.

The final test sequence comprised a slightly larger vehicle travelling at approximately 60 km/h. This test sequence illustrated the tradeoff that is involved in the size of an object and the maximum speed at which it can travel for it to be successfully tracked. In this case, the vehicle was at the ratio's limit and as a result, multiple instances of the vehicle were tracked at the same time. Although this was visually displeasing, a user would still be able to comprehend the movement of the vehicle. Solutions to this problem would be to allow the system operator to increase the size of the motion tracking area or zoom the camera out so that the vehicle would be smaller in size thereby enabling it to travel at higher speeds and still be tracked. The ability for a user to be able to define a motion tracking area dependant on each particular scene was recommended as an improvement to the current system. The average resultant bit-rate of this test sequence was calculated at 7,668 bits/second over the eight second duration.

The performance of the implemented system was compared to the H.263 low bit-rate video compression standard. This was more of a subjective test as the H.263 system was adjusted so that at pre-defined bit-rates, it would operate on exactly the same frames as the implemented system. The results indicated that on a per frame basis in all three test cases, the quality of the decoded video of the implemented system was far superior to the H.263 standard at similar bit-rates.

Whilst the goals of this dissertation have been achieved, it is believed that there are several areas in which the performance of the implemented system could be enhanced, and as a result, several recommendations corresponding to general system improvements and possible new avenues for future research have been identified. An example of a potential algorithmic improvement is to use motion estimation in order to assist object tracking. By doing so, it would be expected that the overall output bit-rate should decrease as instead of retransmitting a complete object each time it had to undergo change, by tracking the individual blocks that make up an object instead, only the blocks that undergo change need to be transmitted. There are many possible alternatives to the algorithms implemented in the current system and it is likely that some may prove more efficient than those used. It has been stressed however, that in the field of image and video compression, it is most often that techniques would have to be experimented with in order to ascertain their suitability to the system at hand. It is therefore believed that due to the basis and structure of its implementation, the current system would provide an adequate starting point for future research projects.

APPENDIX A

COLOUR TRANSFORMATIONS

The following equations have been obtained from [Bhas00]:

A.1 YCbCr

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} . \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad \text{(A-1)}$$

where R, G and B have values in the range [0, 1]

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.000 & 1.402 \\ 1 & -0.344 & -0.714 \\ 1 & 1.771 & 0.000 \end{bmatrix} . \begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} \qquad \text{(A-2)}$$

where Y has a value in the range [0, 1] and Cb and Cr have values in the range [-0.5, 0,5].

A.2 CieLab

The transformation from RGB to CieLab (also known as CieL*a*b*) requires two steps. The first step involves a linear transformation from RGB to Cie XYZ by means of the following:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3575 & 0.1804 \\ 0.2126 & 0.7151 & 0.0721 \\ 0.0193 & 0.1191 & 0.9502 \end{bmatrix} . \begin{bmatrix} R_{709} \\ G_{709} \\ B_{709} \end{bmatrix} \qquad \text{(A-3)}$$

where R, G and B have values in the range [0, 100] as defined by the ITU Recommendation BT.709. The second involves a non-linear transformation in order to obtain the L*a*b* components according to the following:

$$L* = 116\left(\frac{Y}{Y_n}\right) - 16,$$

$$a* = 500\left[f\left(\frac{X}{X_n}\right) - f\left(\frac{Y}{Y_n}\right)\right], \qquad \text{(A-4)}$$

$$b^* = 200\left[f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right)\right]$$

where X_n, Y_n and Z_n are constants defined within the Cie standard and where:

$$f(r) = \begin{cases} r^{\frac{1}{3}} & if \quad r > 0.008856 \\ 7.7867r + \frac{16}{116} & otherwise \end{cases}$$

where L* is the luminance component that has a value in the range [0, 100] and a* and b* are the two chrominance components that a have values in the range [-100, 100].

RGB can be obtained from CieL*a*b* by application of the following:

If L* < 8.0, then

$$\frac{Y}{Y_n} = \frac{L^*}{903.3}$$

else

$$\frac{Y}{Y_n} = \frac{1}{100}\left[\frac{L^*+16}{25}\right]^3 \qquad \text{(A-5)}$$

Given Y,

$$\frac{X}{X_n} = \left[\frac{a^*}{500} + \left(\frac{Y}{Y_n}\right)^{\frac{1}{3}}\right]^3$$

$$\frac{Z}{Z_n} = \left[\left(\frac{Y}{Y_n}\right)^{\frac{1}{3}} - \frac{b^*}{200}\right]^3$$

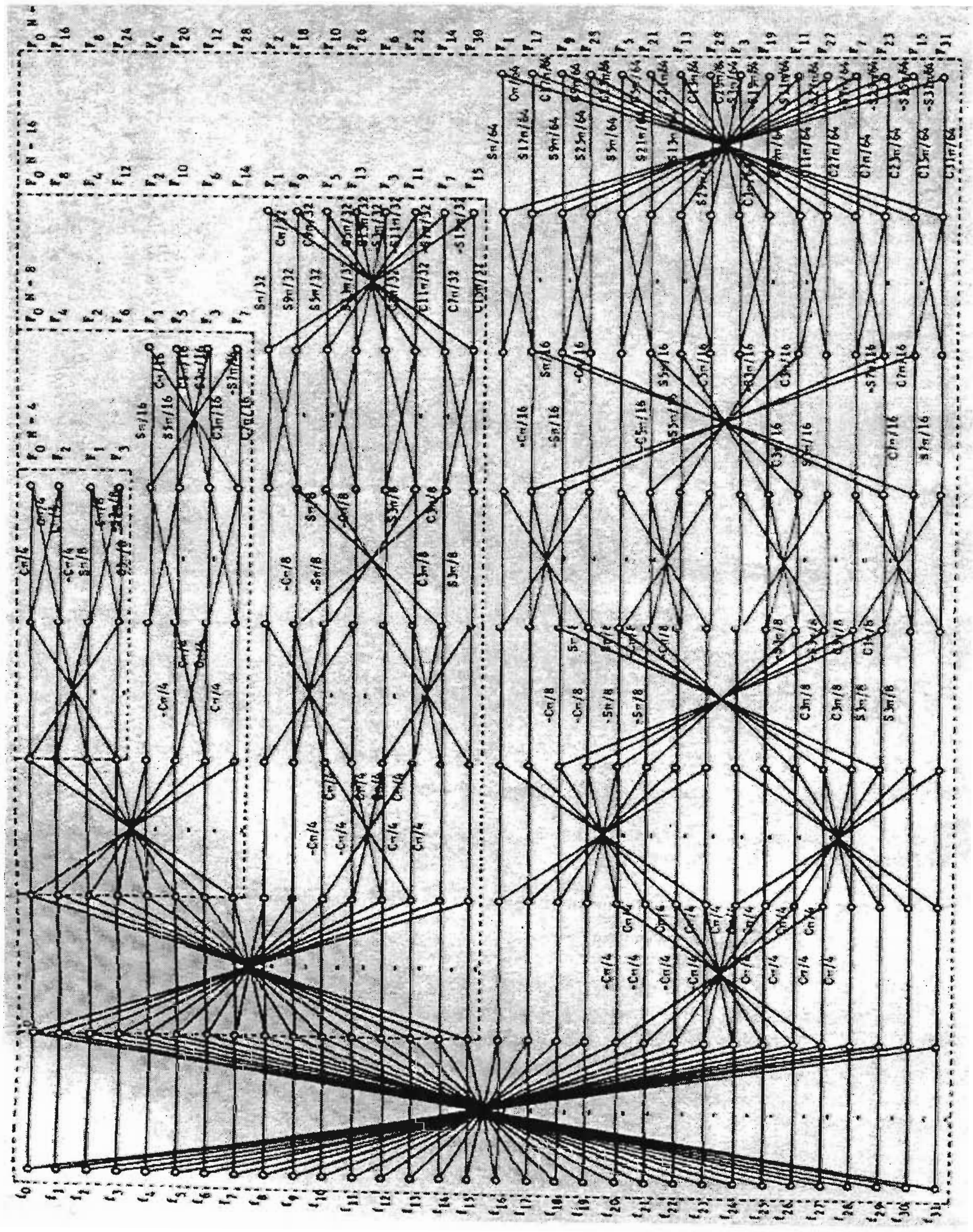
and finally

$$\begin{bmatrix} R_{709} \\ G_{709} \\ B_{709} \end{bmatrix} = \begin{bmatrix} 3.2404 & -1.5371 & -0.4985 \\ -0.9692 & 1.8759 & 0.0415 \\ 0.0556 & -0.2040 & 1.0573 \end{bmatrix} . \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \qquad \text{(A-6)}$$

APPENDIX B

FAST 1D DISCRETE COSINE TRANSFORM

The following signal flow chart has been obtained from [Chen77]:

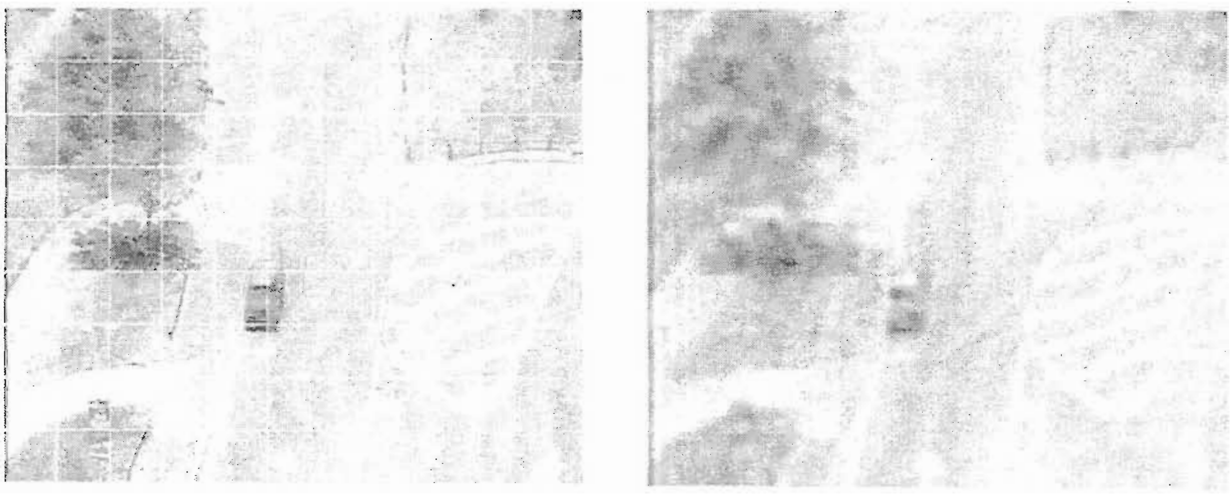


APPENDIX C

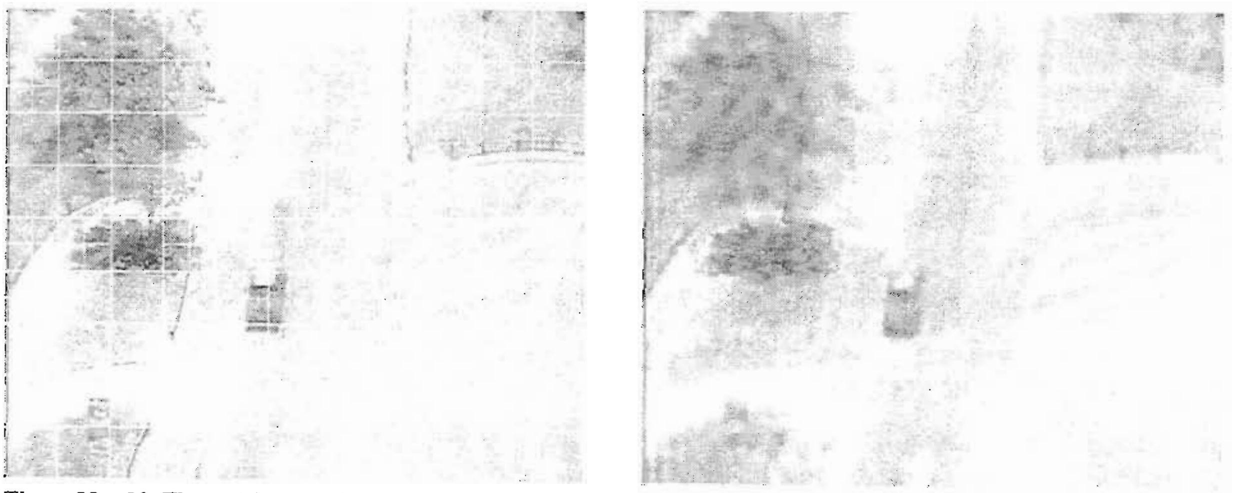
VIDEO FRAME TEST RESULTS

C.1 Decoded Results of VF-2:

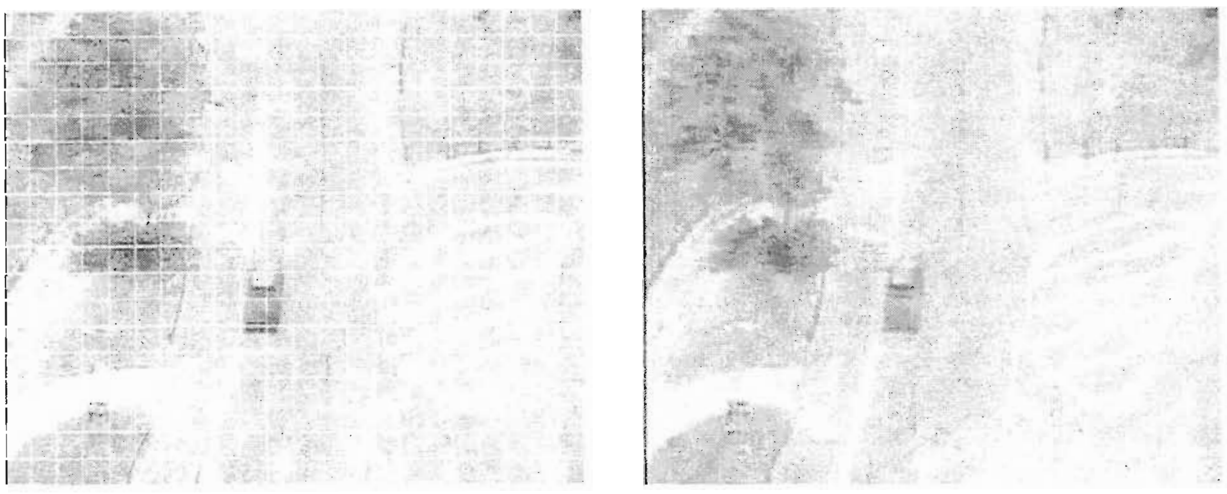
The following results were obtained by varying Thres_32 and Thres_16 independently of each other. The variable settings and results obtained are displayed below each set of video frames. Each set comprises the variable block-size sub-divided original test video frame on the left side with its decoded counterpart on the right:

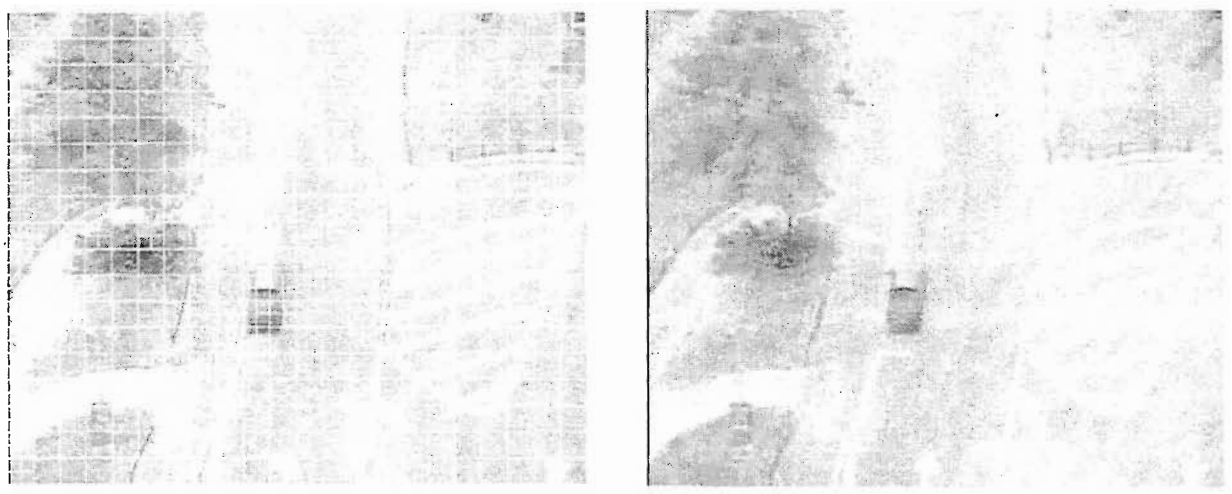


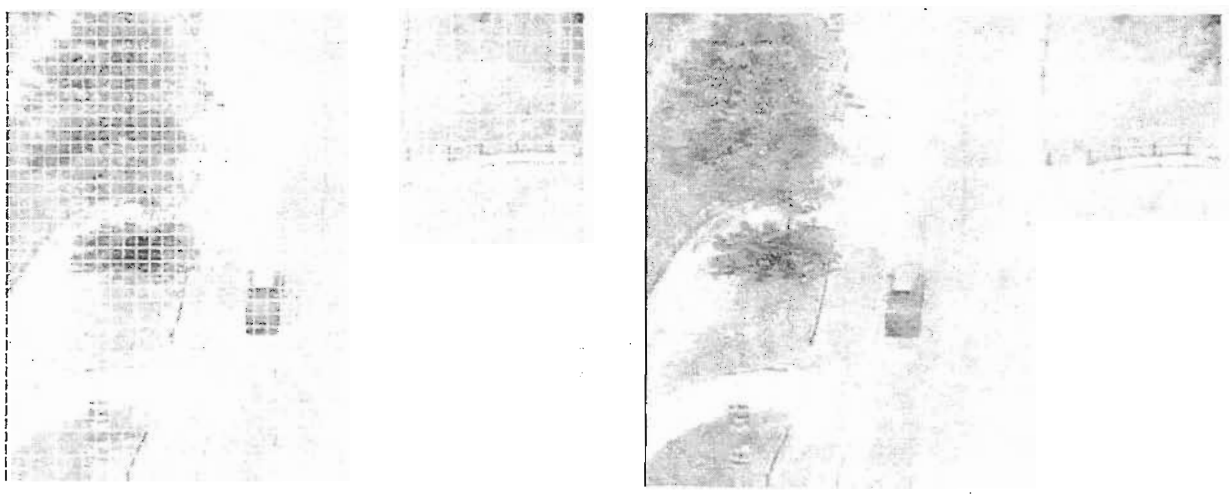
Thres_32 = 70, **Thres_16** = 200, **Compression Ratio** = 52.54, **bpp** = 0.1523, **Y PSNR** = 27.61 dB

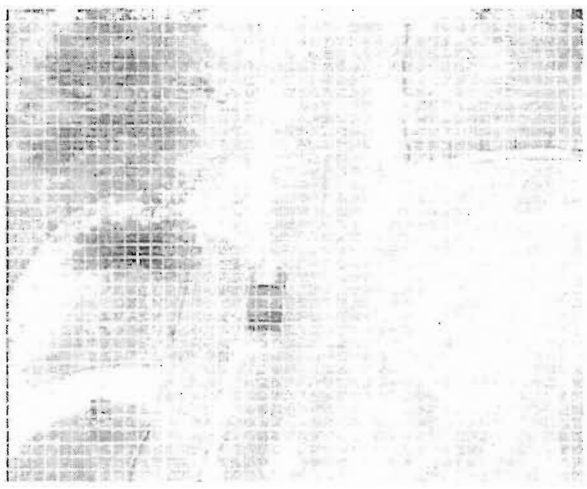


Thres_32 = 30, **Thres_16** = 200, **Compression Ratio** = 48.68, **bpp** = 0.1643, **Y PSNR** = 28.39 dB


Thres_32 = 0, **Thres_16** = 200, **Compression Ratio** = 27.93, **bpp** = 0.2864, **Y PSNR** = 31.19 dB

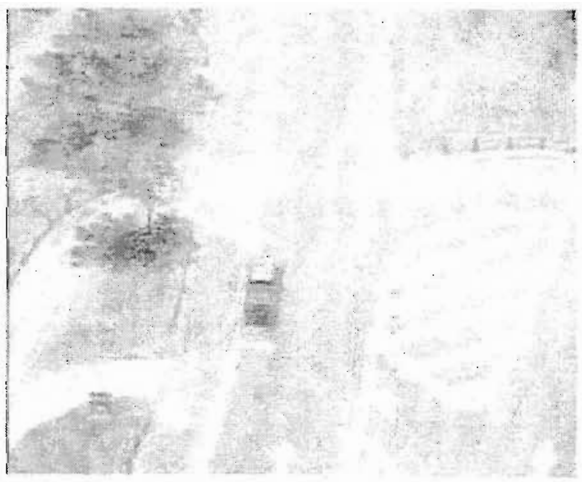

Thres_32 = 0, **Thres_16** = 30, **Compression Ratio** = 25.6, **bpp** = 0.3125, **Y PSNR** = 32.31 dB


Thres_32 = 0, **Thres_16** = 10, **Compression Ratio** = 18.57, **bpp** = 0.4308, **Y PSNR** = 35.01 dB

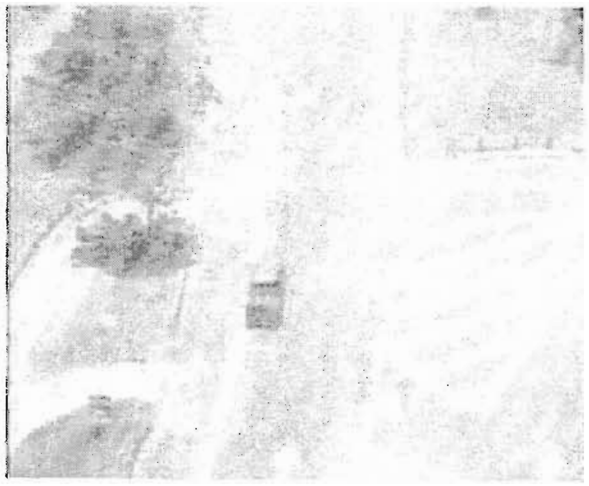


Thres_32 = 0, **Thres_16** = 0, **Compression Ratio** = 15.54, **bpp** = 0.5148, **Y PSNR** = 35.28 dB

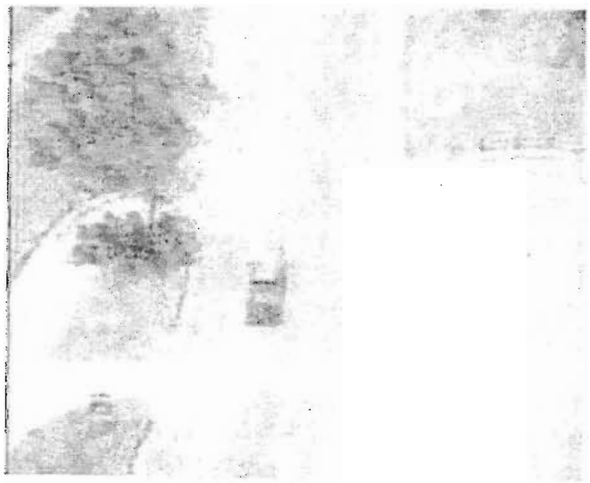
The following decoded test video frames of VF-2 were obtained by varying **Thres_32** and **Thres_16** simultaneously in relation to each other. The variable settings and measured results are displayed below each decoded video frame:



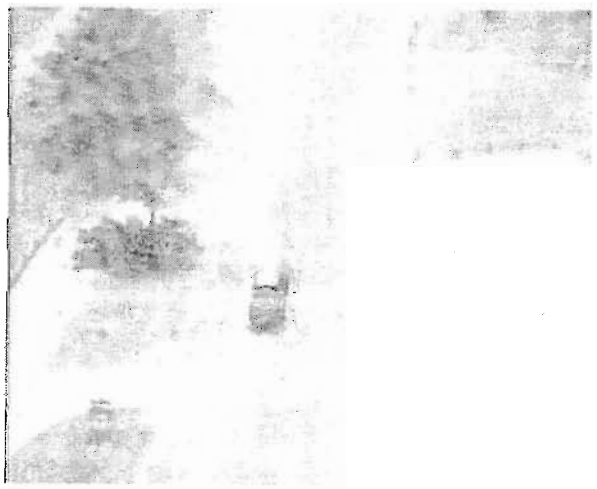
Thres_32 = 15, **Thres_16** = 30, **CR** = 32.68, **bpp** = 0.2448, **Y PSNR** = 31.52 dB



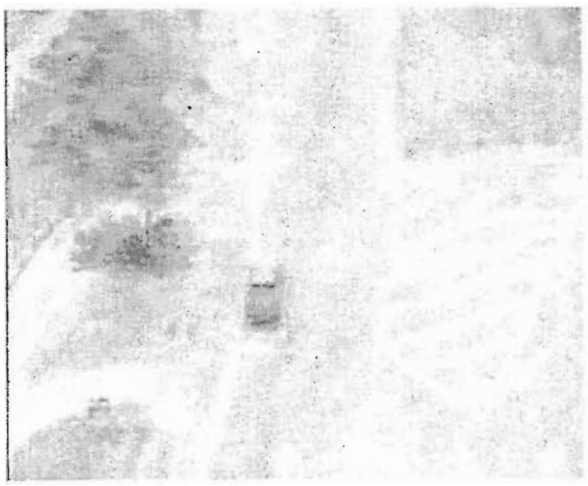
Thres_32 = 15, **Thres_16** = 40, **CR** = 21.90, **bpp** = 0.2190, **Y PSNR** = 30.60 dB



Thres_32 = 15, **Thres_16** = 50, **CR** = 36.74, **bpp** = 0.2178, **Y PSNR** = 30.57 dB



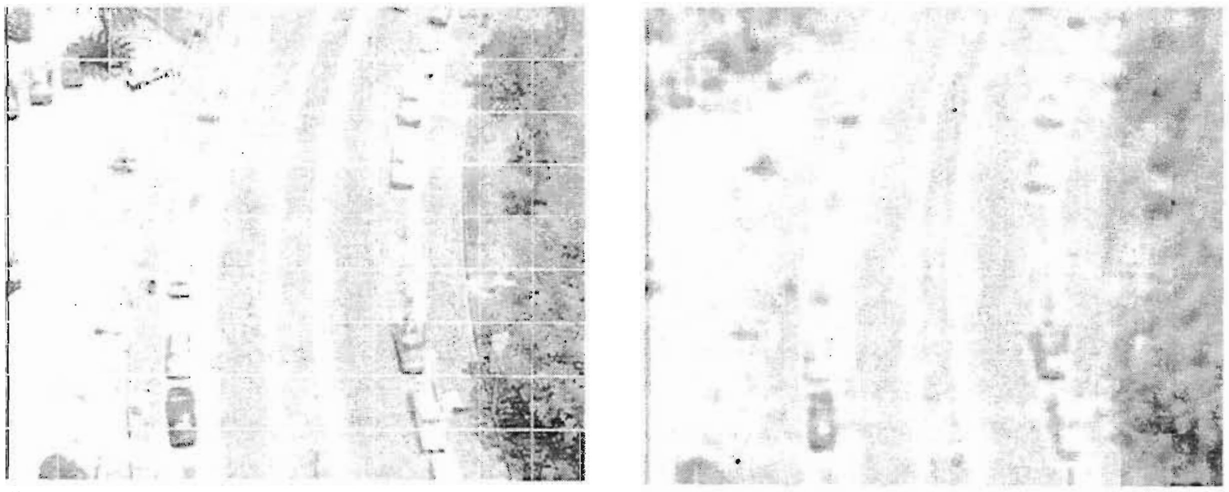
Thres_32 = 25, **Thres_16** = 30, **CR** = 38.50, **bpp** = 0.2078, **Y PSNR** = 30.11 dB

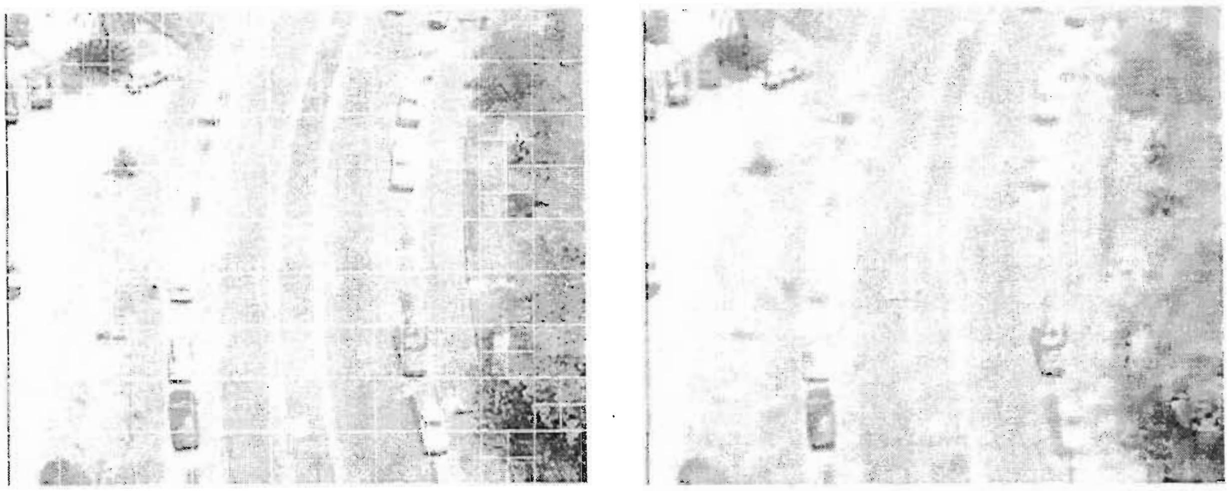

Thres_32 = 25, **Thres_16** = 40, **CR** = 43.17, **bpp** = 0.1853, **Y PSNR** = 29.49 dB

Thres_32 = 25, **Thres_16** = 50, **CR** = 43.46, **bpp** = 0.1841, **Y PSNR** = 29.46 dB

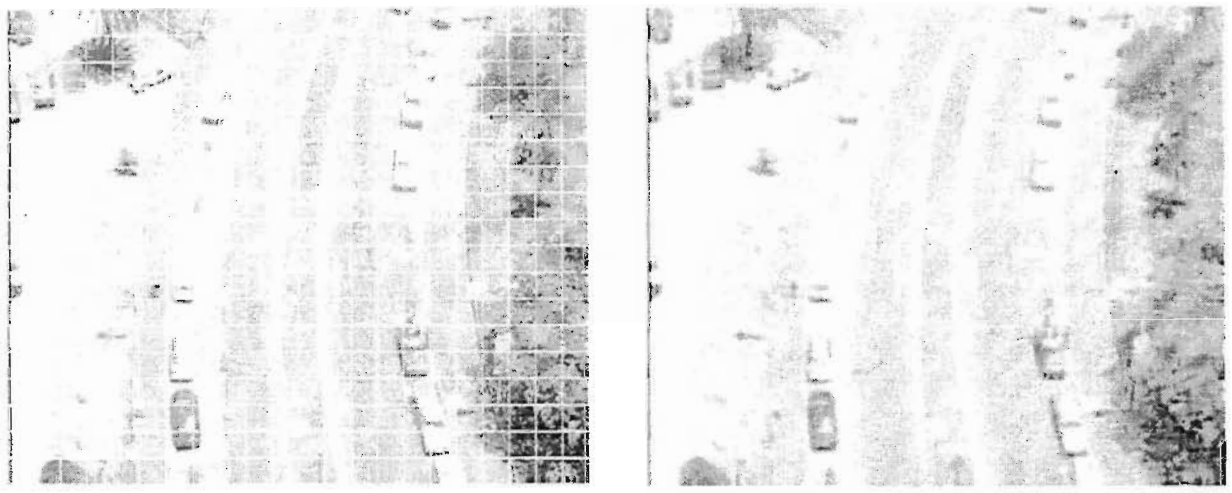
C.2 Decoded Results of VF-3:

The following results were obtained by varying **Thres_32** and **Thres_16** independently of each other. The variable settings and results obtained are displayed below each set of video frames. Each set comprises the variable block-size sub-divided original test video frame on the left side with its decoded counterpart on the right:

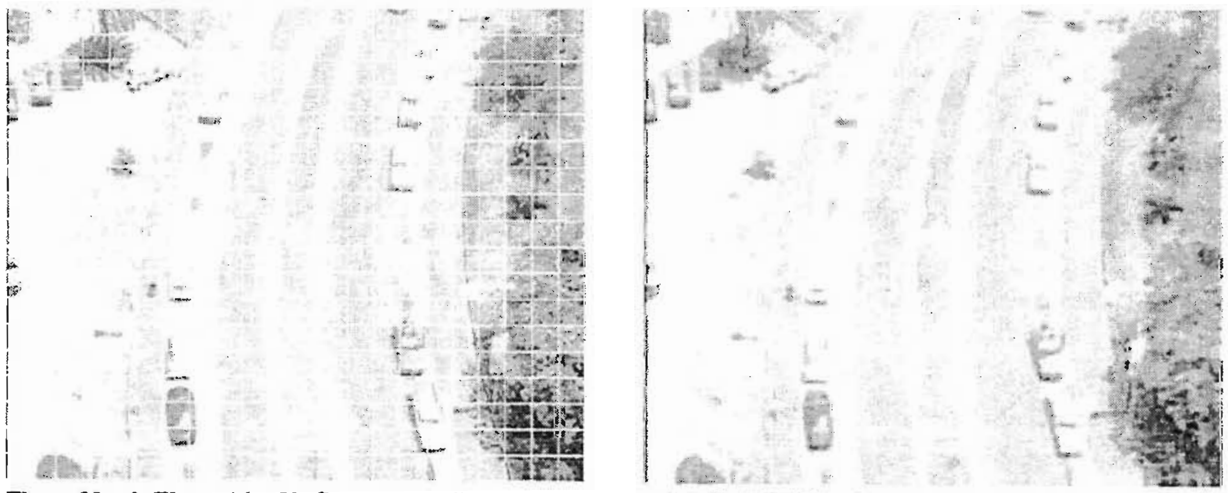

Thres_32 = 70, **Thres_16** = 200, **Compression Ratio** = 48.8, **bpp** = 0.1639, **Y PSNR** = 25.2 dB



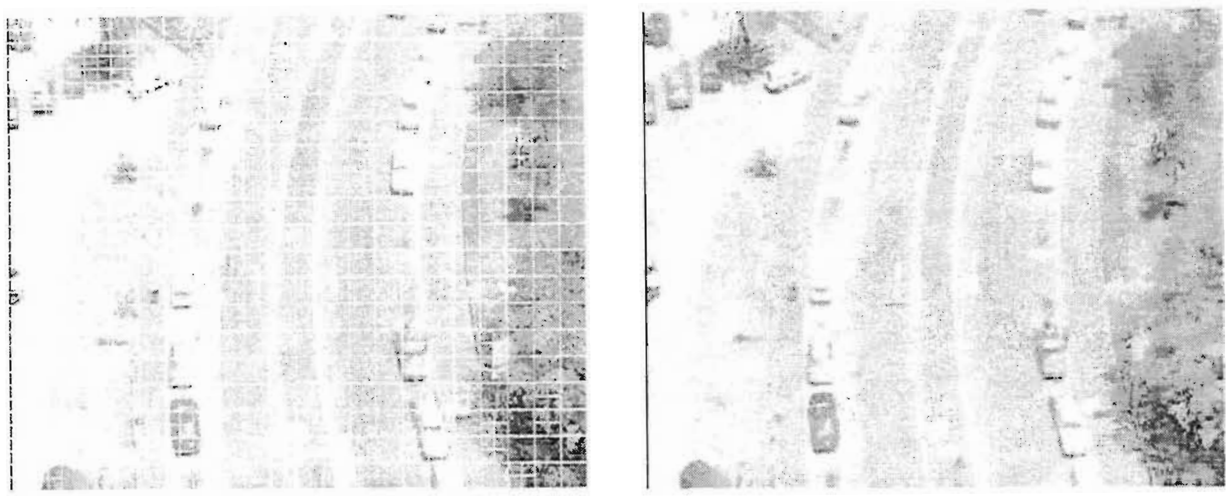
Thres_32 = 30, **Thres_16** = 200, **Compression Ratio** = 40.05, **bpp** = 0.1998, **Y PSNR** = 27.2 dB

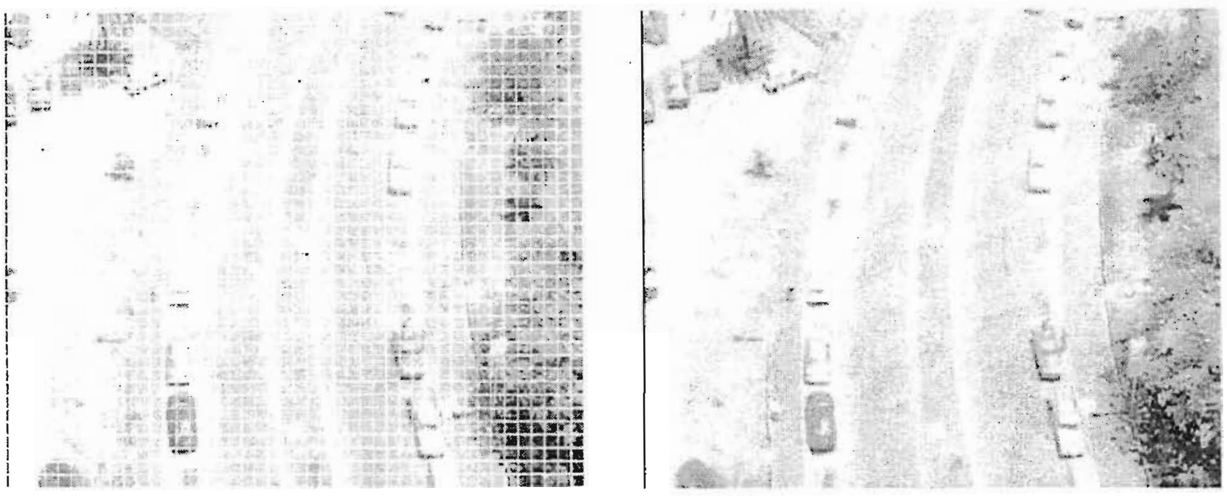


Thres_32 = 0, **Thres_16** = 200, **Compression Ratio** = 25.28, **bpp** = 0.3164, **Y PSNR** = 29.63 dB

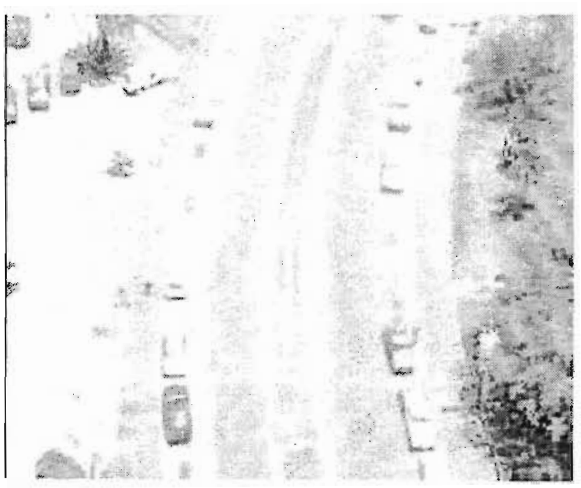


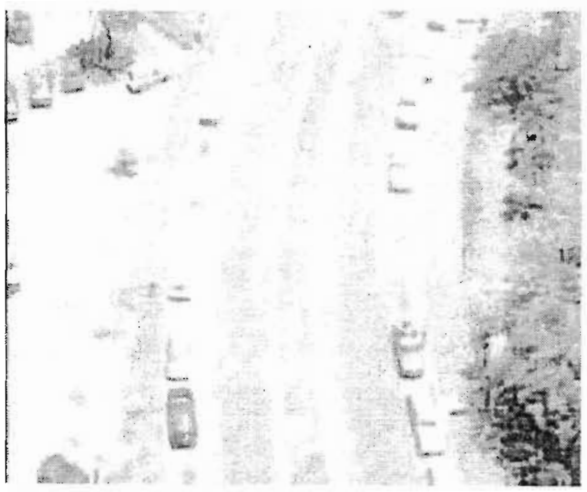
Thres_32 = 0, **Thres_16** = 50, **Compression Ratio** = 25.06, **bpp** = 0.3193, **Y PSNR** = 29.75 dB

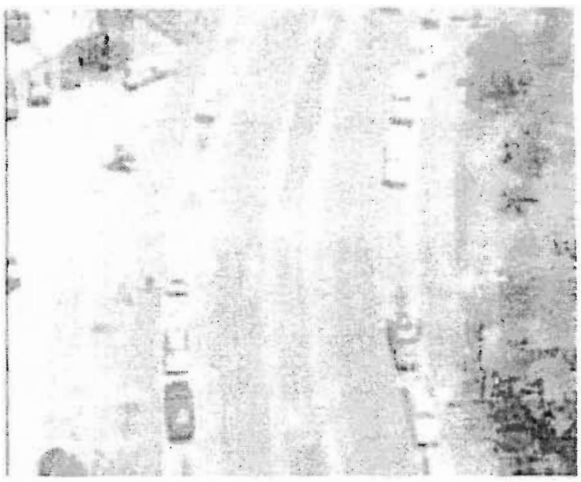

Thres_32 = 0, **Thres_16** = 30, **Compression Ratio** = 21.52, **bpp** = 0.3718, **Y PSNR** = 31.42 dB

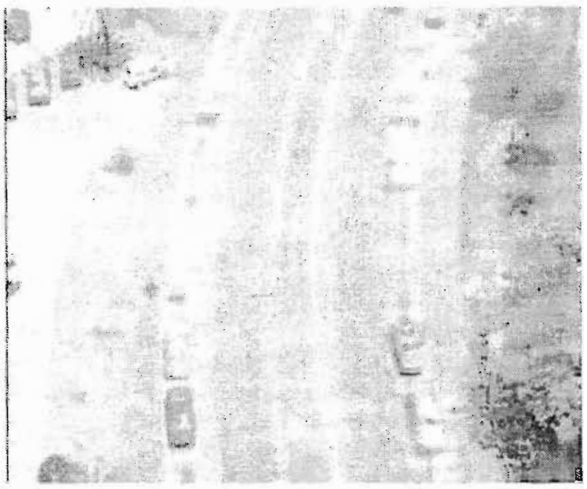

Thres_32 = 0, **Thres_16** = 30, **Compression Ratio** = 13.91, **bpp** = 0.5750, **Y PSNR** = 34.77 dB

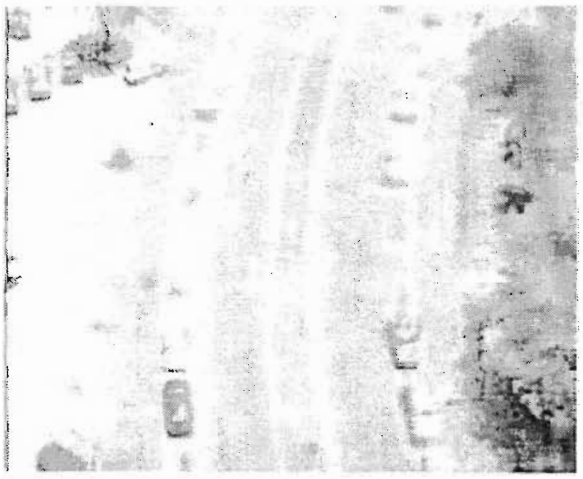
The following decoded test video frames of VF-3 were obtained by varying **Thres_32** and **Thres_16** simultaneously in relation to each other. The variable settings and measured results are displayed below each decoded video frame:

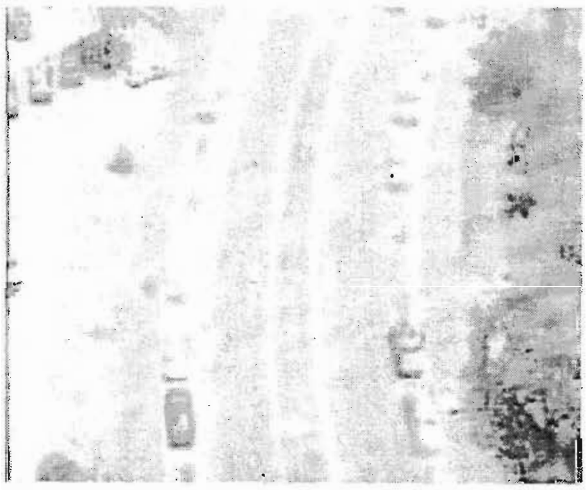

Thres_32 = 15, **Thres_16** = 30, **CR** = 25.02, **bpp** = 0.3198, **Y PSNR** = 30.79 dB


Thres_32 = 15, **Thres_16** = 40, **CR** =28.36, **bpp** = 0.2821, **Y PSNR** = 29.71 dB


Thres_32 = 15, **Thres_16** = 50, **CR** = 30.09, **bpp** = 0.2659, **Y PSNR** = 29.31 dB


Thres_32 = 25, **Thres_16** = 30, **CR** = 29.95, **bpp** = 0.2671, **Y PSNR** = 29.00 dB


Thres_32 = 25, **Thres_16** = 40, **CR** = 33.71, **bpp** = 0.2373, **Y PSNR** = 28.33 dB

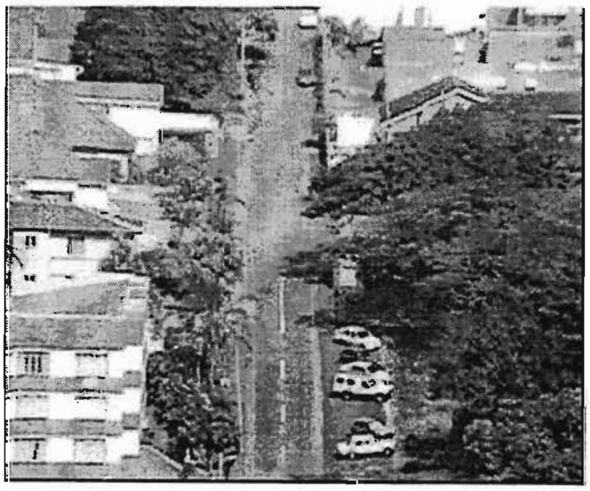

Thres_32 = 25, **Thres_16** = 50, **CR** = 36.22, **bpp** = 0.2209, **Y PSNR** = 28.04 dB

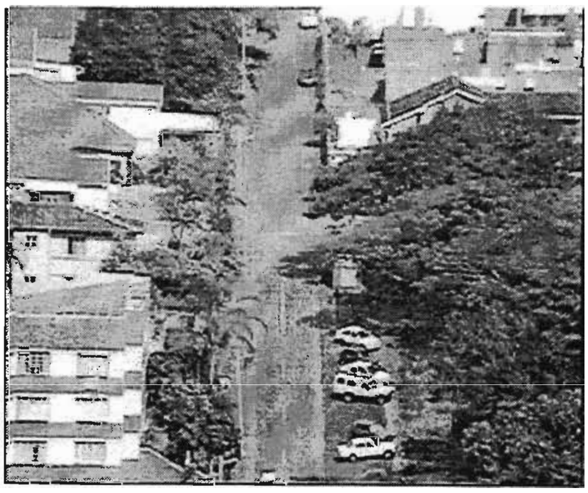
APPENDIX D

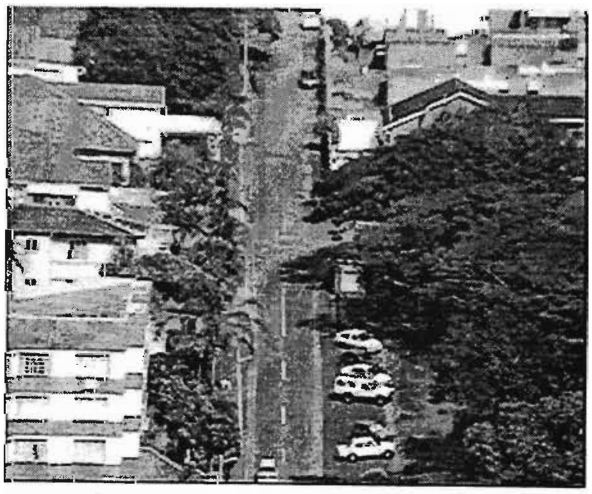
TEST VIDEO SEQUENCES

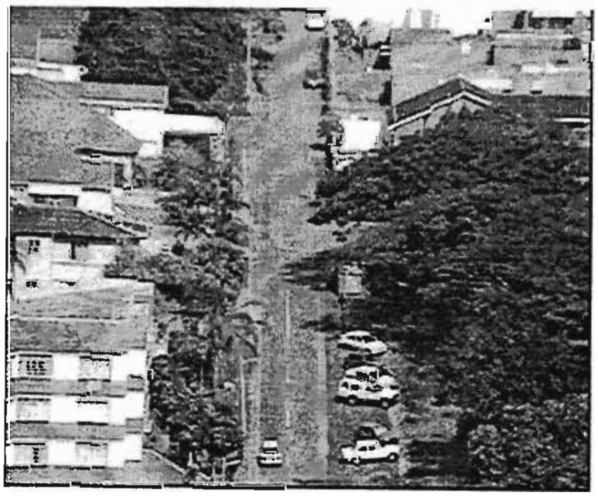
The following target video frames form the three test video sequences that were used to evaluate the performance of the implemented object segmentation and tracking algorithms.

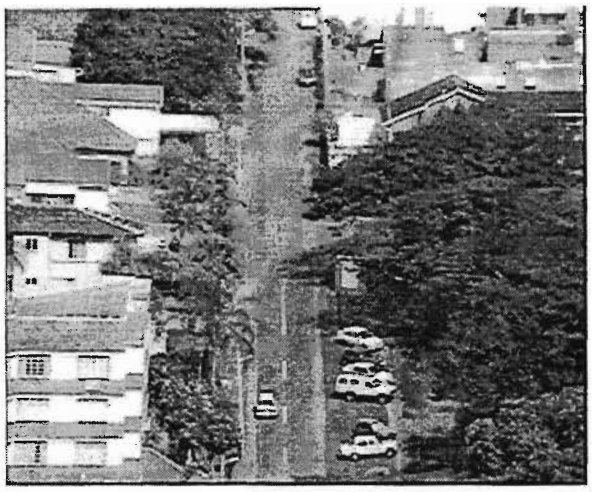
D.1 Test Video Sequence VS-1:

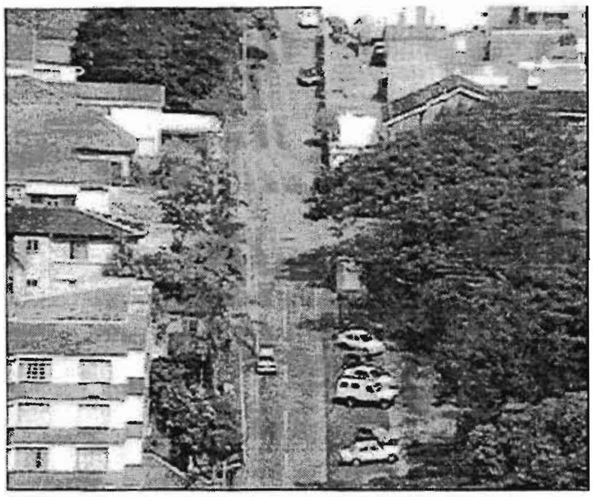

target video frame: 0

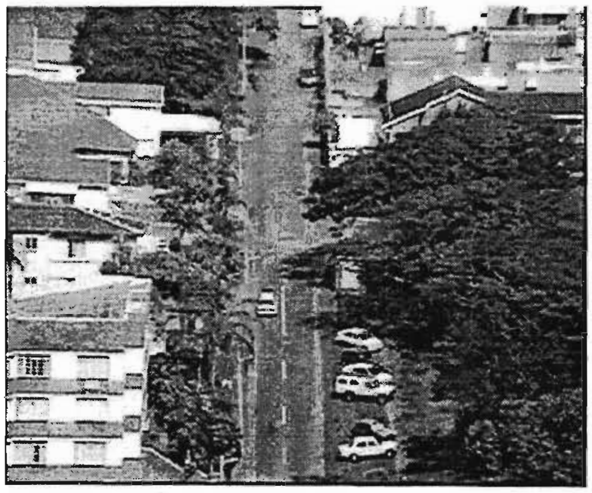

target video frame: 4

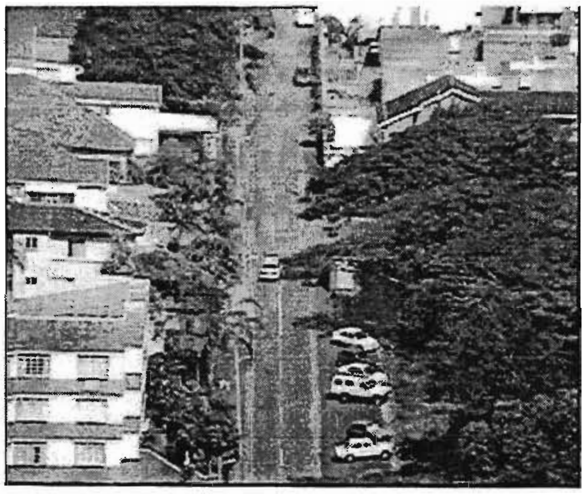

target video frame: 6


target video frame: 9

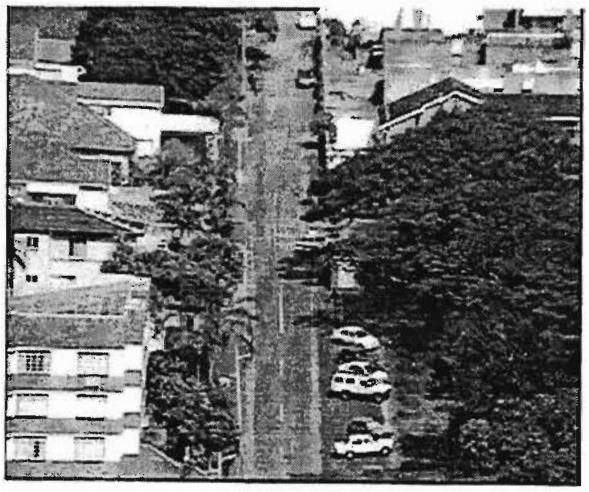

target video frame: 14

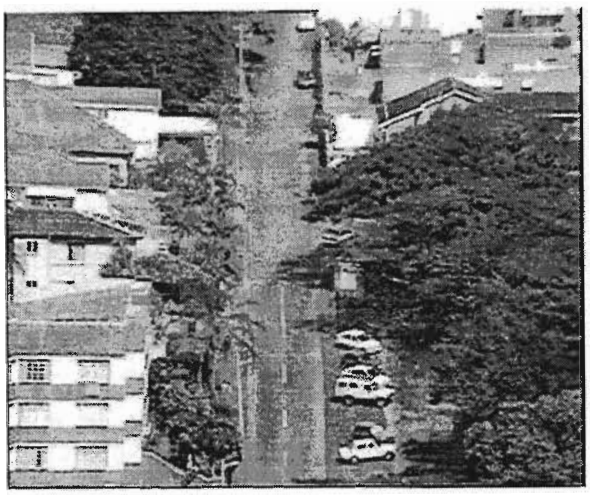

target video frame: 18


target video frame: 24


target video frame: 28

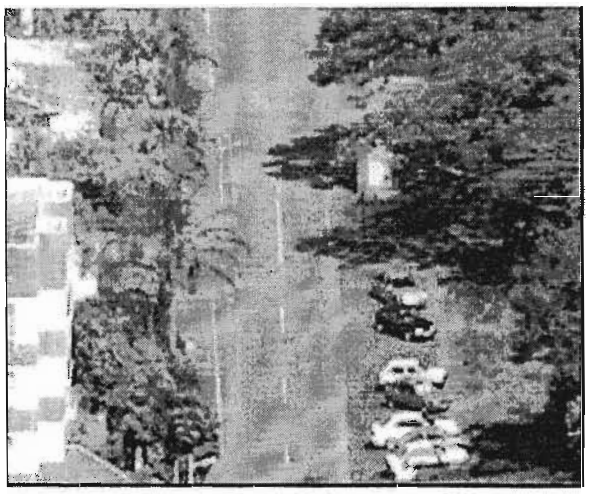
target video frame: 32

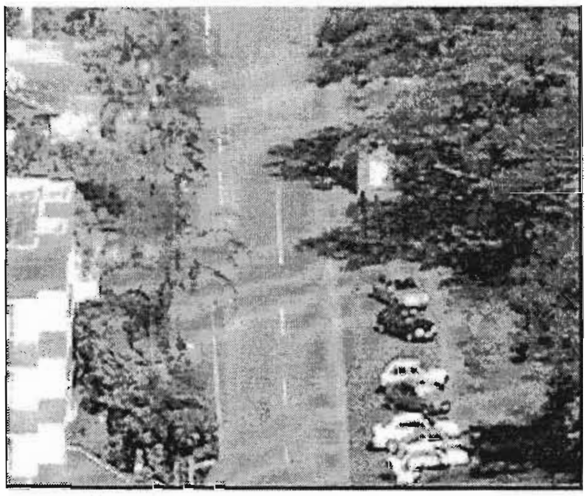

target video frame: 34

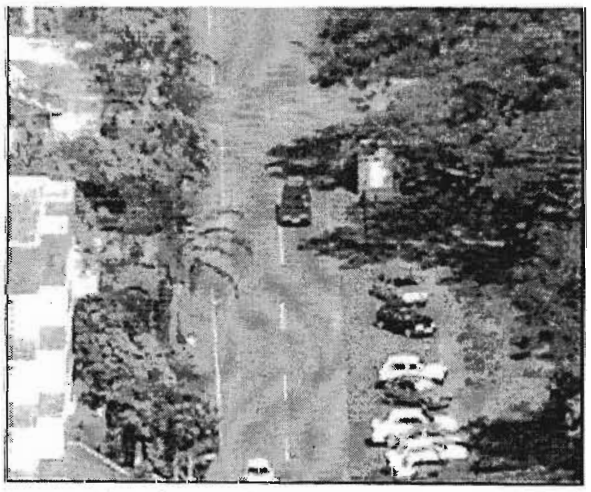

target video frame: 35

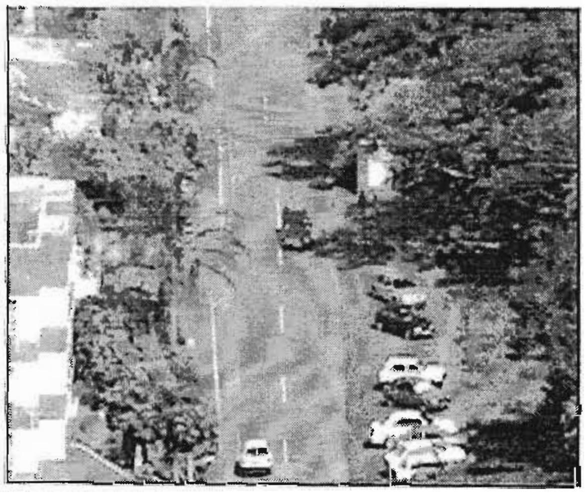
target video frame: 38

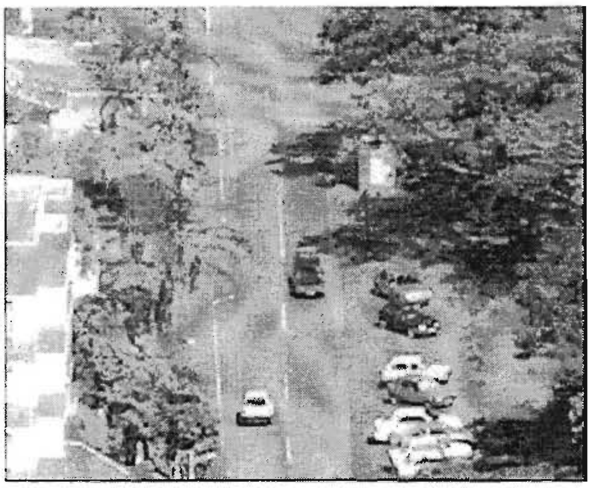
D.2 Test Video Sequence VS-2:

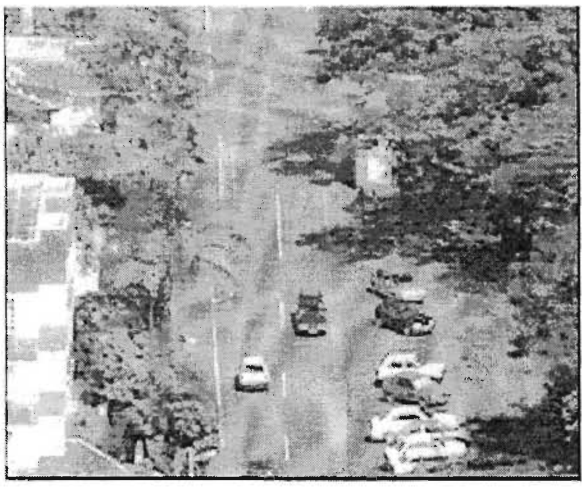

target video frame: 0

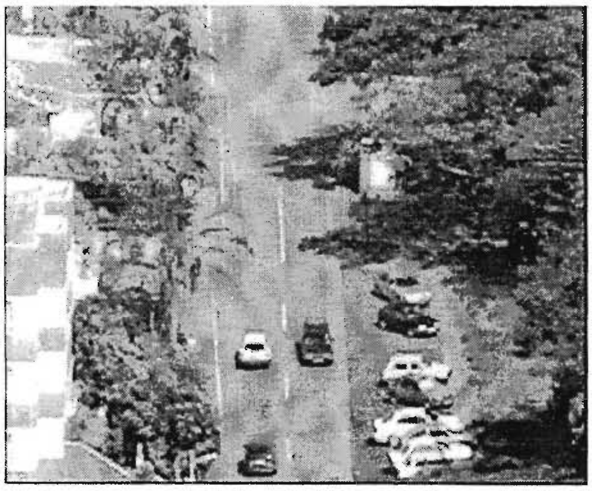

target video frame: 3

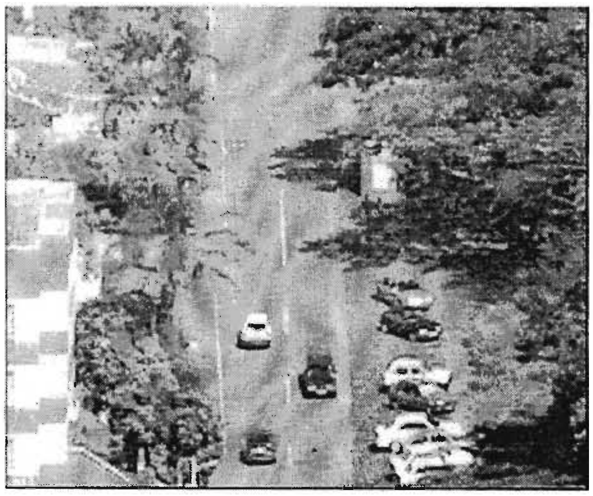

target video frame: 6

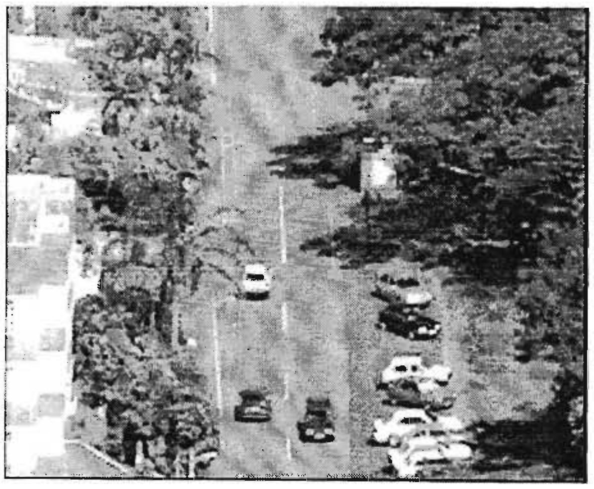

target video frame: 7

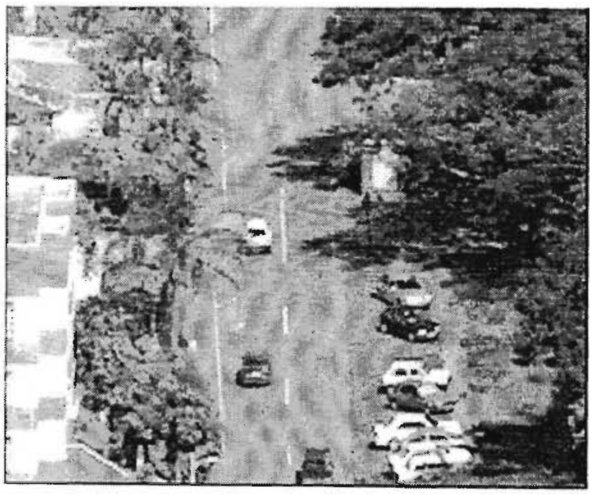

target video frame: 9

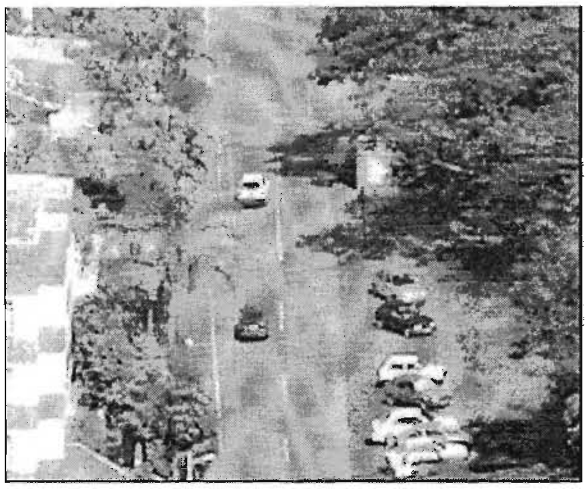

target video frame: 10


target video frame: 11

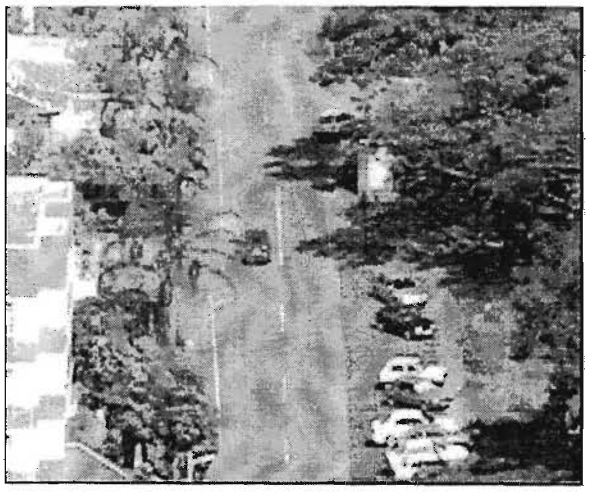

target video frame: 12

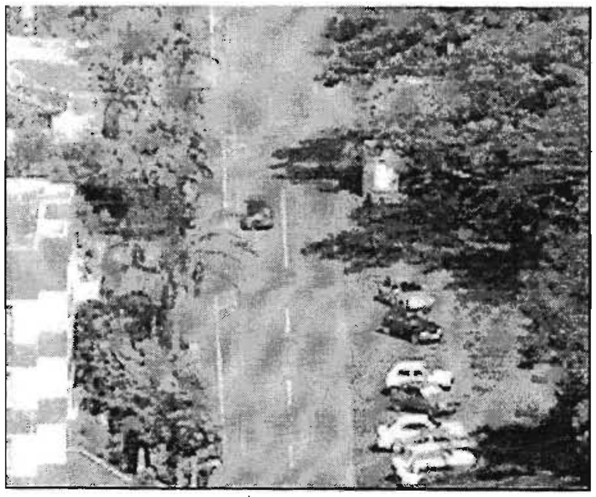

target video frame: 14

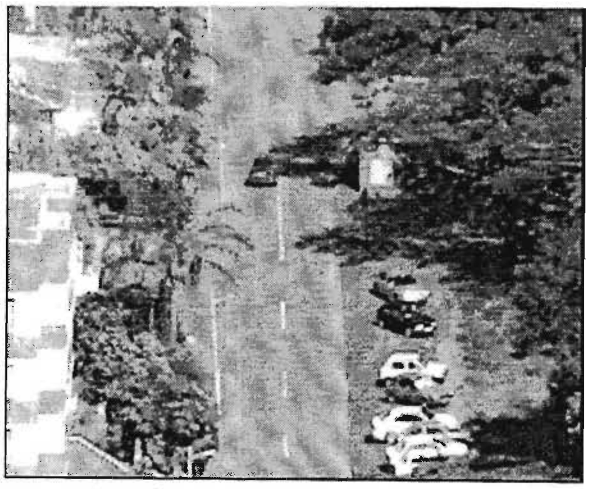

target video frame: 16

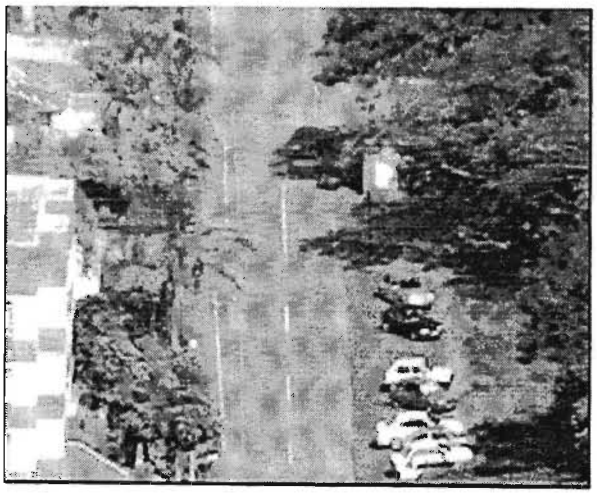

target video frame: 18

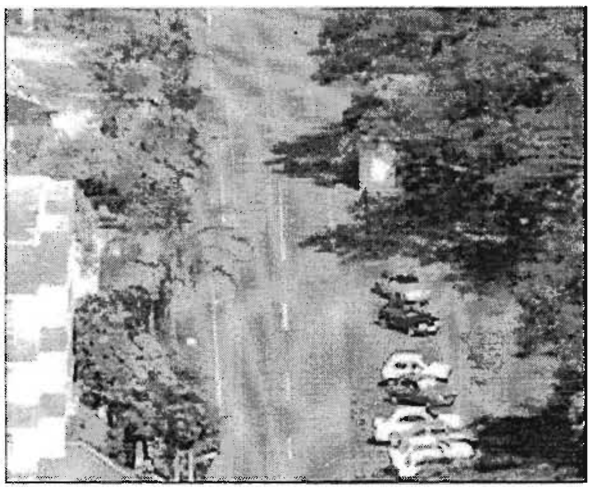
target video frame: 20


target video frame: 22


target video frame: 24


target video frame: 26


target video frame: 28


target video frame: 30

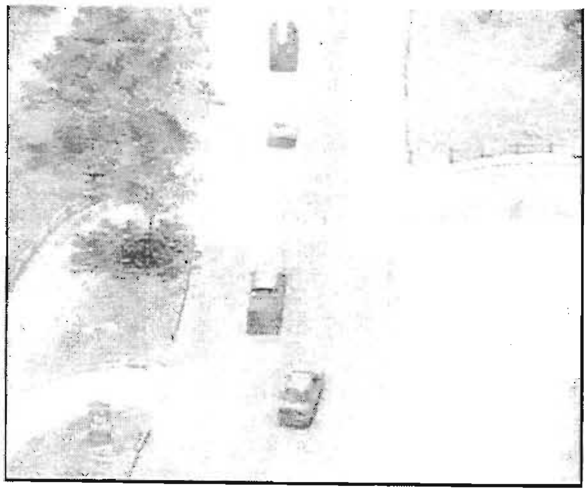
target video frame: 31

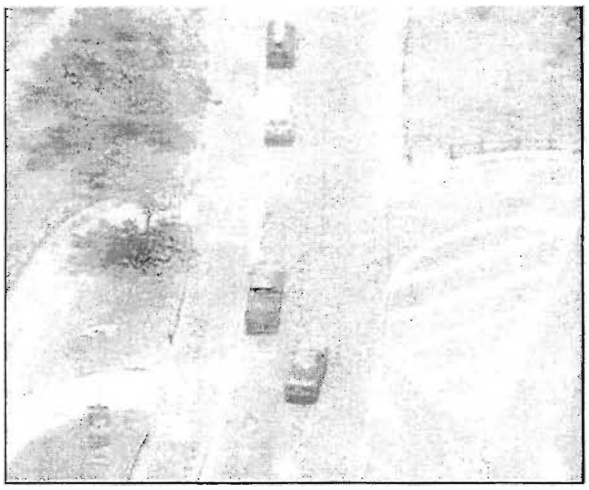
D.3 Test Video Sequence VS-3:

target video frame: 0

target video frame: 2

target video frame: 4


target video frame: 5

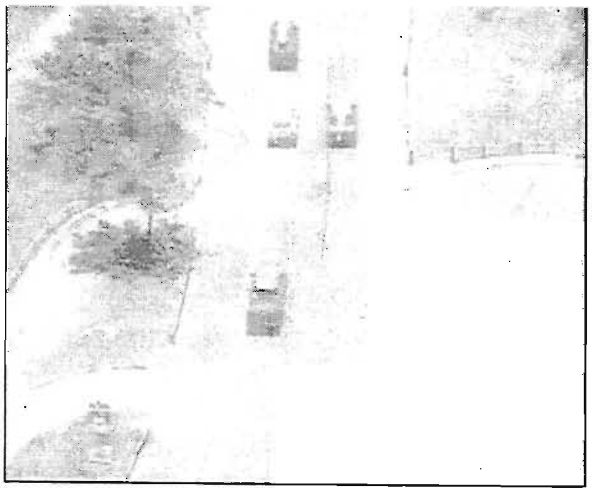

target video frame: 6

target video frame: 7

target video frame: 8

target video frame: 9

target video frame: 11


target video frame: 12

target video frame: 13

target video frame: 14

target video frame: 15

target video frame: 16

target video frame: 17

target video frame: 18

REFERENCES

REFERENCES

[Adams94], Adams, R., Bischof, L., "Seeded Region Growing", IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 641-647, June 1994.

[Ahmed74], Ahmed, N., Natarjan, T., Rao, K. R., "Discrete Cosine Transform", IEEE Transactions on Computers, vol C-32, pp. 90-93, January 1974.

[Berc98], Berc, L., Fenner, W., Frederick, R., McCanne, S., Stewart, P., "RTP Payload Format for JPEG-compressed Video", 1998, http://community.roxen.com/developers/idocs/rfc/rfc2435.html.

[Bhas00], Bhaskaran, V., Konstantinides, K., "Image and Video Compression Standards", Kluwer Academic Publishers, 2nd Edition, 2000.

[Bourke94], Bourke, P., "Colour space conversion", February 1994, http://astronomy.swin.edu.au/pbourke/colour/conversion.html.

[Brem98], Bremond, F., Thonnat, M., "Tracking Multiple Nonrigid Objects in Video Sequences", IEEE Transactions on Circuits and Systems for Video Technology, pp. 585-592, September 1998.

[Calway00], Calway, A., "Camera and Structure Model", 2000, http://www.cs.bris.ac.uk/~andrew/3dsm/node2.html.

[Canny86], Canny, J., "A Computational Approach to Edge Detection", IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 679-698, November 1986.

[Celasun01], Celasun, I., Tekalp, A. M., Gokcetekin, M. H., Harmanci, D. M., "2-D Mesh-Based Video Object Segmentation and Tracking with Occlusion Resolution", Signal Processing: Image Communication, vol. 16, no. 10, pp. 949-962, August 2001.

[Cerveri98], Cerveri, P., Pinciroli, F., "Multiresolution Image Representation Through Wavelet Compression for Speeding Up Navigation in the Visible Human Data Set Archive", 1998, http://www.nlm.nih.gov/research/visible/vhpconf98/AUTHORS/CERVERI/CERVERI2/CERVERI2.HTM.

[Chen77], Chen, W-H., Smith, C. H., Fralick, S. C., "A Fast Computational Algorithm for the Discrete Cosine Transform", IEEE Transactions on Communications, pp. 1004-1009, September 1977.

[Chen77(2)], Chen, W-H., Smith, "Adaptive Coding of Monochrome and Color Images", IEEE Transactions on Communications, pp. 1285-1292, November 1977.

[Cho91], Cho, N. I., Lee, S. U., "Fast Algorithm and Implementation of 2-D Discrete Cosine Transform", IEEE Transactions on Circuits and Systems, pp. 259-266, March 1991.

[Damer00], Damera-Venkata, N., Kite, T., Geisler, W., Evans, B., Bovik, A., "Image Quality Assessment Based on a Degradation Model", IEEE Transactions on Image Processing, pp. 636-650, April 2000.

[Digital01], CSIRO Australia, "Digital Video Formats – facts and figures", http://www.cmis.csiro.au/DMIS/VideoTalk/vformat.html, 2001.

[Dins90], Dinstein, I., Rose, K., Heiman, A., "Variable Block-Size Transform Image Coder", IEEE Transactions on Communications, pp. 94-101, November 1990.

[Docef02], Docef, A., Kossentini, F., Nguuyen-Phi, K., Ismeil, I., "The Quantised DCT and its Application to DCT-Based Video Coding", IEEE Transactions on Image Processing, pp. 177-187, March 2002.

[Done96], Done, S., Basith, S., "Digital Video, MPEG and Associated Artifacts", June 1996, http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/sab/report.html.

[Ebrah00], Touradj, E., Horne, C., "MPEG-4 Natural Video Coding – An Overview", 2000, http://leonardo.telecomitalialab.com/icjfiles/mpeg-4_si/7-natural_video_paper/7-natural_video_paper.htm.

[Elliot82], Elliot, D. F., Rao, K. R., "Fast Transforms: Algorithms, Analyses, Applications.", Academic Press, 1982.

[Everitt93], Everitt, B., "Cluster Analysis", Halstead Press NY, 2nd Edition, 1993.

[Fieg92], Feig, E., Winograd, S., "Fast Algorithms for the Discrete Cosine Transform", IEEE Transactions on Signal Processing, pp. 2174-2193, September 1992.

[Gallager78], Gallager, R. G., "Variations on a Theme by Huffman", IEEE Transactions on Information Theory, pp. 668-674, November 1978.

[Gera94], Gerald, C. F., Wheatley, P. O., "Applied Numerical Analysis", Addison Wesely, 5th Edition, 1994.

[Gerf96], Gerfelder, N., Muller, W., "Objective Quality Estimation for Digital Images and Video Streams", published in ITG, EURASIP, "Workshop on Quality Assessment in Speech, Audio and Image Communication", pp. 71-87, March 1996, http://citeseer.nj.nec.com/37177.html.

[Gimlett75], Gimlett, J. I., "Use of "Activity" Classes in Adaptive Transform Image Coding", IEEE Transactions on Communications, pp. 785-786, July 1975.

[Gu98], Gu, C., Lee, M-C, "Semiautomatic Segmentation and Tracking of Semantic Video Objects", IEEE Transactions on Circuits and Systems for Video Technology, pp. 572-584, September 1998.

[Gunsel98], Gunsel, B., Ferman, A. M., Tekalp, A. M., "Temporal Video Segmentation using Unsupervised Clustering and Semantic Object Tracking", Journal of Electronic Imaging 7(3), pp. 592-604, July 1998, http://citeseer.nj.nec.com/gunsel98temporal.html

[H.263], "Video Coding for Low Bit Rate Communication", http://www.cmlab.csie.ntu.edu.tw/cml/dsp/training/coding/h263/welcome.html.

[Hart75], Hartigan, J. A., "Clustering Algorithms", John Wiley & Sons NY, 1st Edition, 1975.

[Han98], Han, S-C., Woods, J. W., "Adaptive Coding of Moving Objects for Very Low Bit-Rates", IEEE Journal on Selected Areas in Communication, pp. 56-61, January 1998.

[Hontsch02], Hontsch, I., Karam, L., "Adaptive Image Coding with Perceptual Distortion Control", IEEE Transactions on Image Processing, pp. 213-222, March 2002.

[Howard92], Howard, P. G., Vitter, J. S., "Analysis of Arithmetic Coding for Data Compression", Information Processing and Management, vol. 28, number 6, pp. 749-763, November 1992.

[Huffman52], Huffman, D. A., "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the I.R.E., pp. 1098-1101, September 1952.

[Hutten93], Huttenlocher, D. P., Klanderman, G. A., Rucklidge, W. J., "Comparing Images Using the Hausdorff Distance", IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 850-863, September 1993.

[InformIT01], Gregory, K., "Special Edition Using Visual C++ ver. 6", last referenced September 2001, http://www.informit.com/content/articlex.asp?product_id={31F9A400-859D-4FBA-B14A-AC4DAE32011C}&element_id={BEF354E1-2FCE-4A14-A153-6FB7BD738A33}&session_id={237CA89E-F991-405E-B8AF-FA477912DBB7}&st=FAA431F7-11F1-4FBF-8681-0C84321AEB69.

[Jack01], Jack, K., "Video Demystified", LLH Technology Publishing, 2001.

[Jain81], Jain, J. R., Jain, A. K., "Displacement Measurement and Its Application in Interframe Image Coding", IEEE Transactions on Communications, pp. 1799-1808, December 1981.

[Jain99], Jain, A. K., Murty, M. N., Flynn, P. J., "Data Clustering: A Review", ACM Computing Surveys, Vol 31, No. 3, pp. 264-323, September 1999, http://www.chipcenter.com/exittracking.dyn?path=http%3A%2F%2Fwww.amk.alt-neustadt.at%2Fdiplom%2Fpapers%2FClustering%2Fp264-jain.pdf.

[JPEG01], "Home site of the JPEG and JBIG committees", last referenced August 2001, http://www.jpeg.org/.

[JPEG(2)99], "JPEG image compression FAQ, part 1 / 2", Subject [20], March 1999, http://www.faqs.org/faqs/jpeg-faq/part1/index.html.

[Kim00], Kim, C., Hwang, J-N., "An Integrated Scheme for Object-based Video Abstraction", 2000, http://students.washington.edu/cikim/cidil/va/acm2000final9.pdf.

[Knuth85], Knuth, D. E., "Dynamic Huffman Coding", Journal of Algorithms 6, pp. 163-180, 1985.

[Koller93], Koller, D., Weber, J., Malik, J., "Robust Multiple Car Tracking with Occlusion Reasoning", Technical Report UCB/CSD 93/780, Computer Science Division (EECS), University of California, Berkeley, 1993, http://citeseer.nj.nec.com/koller93robust.html.

[Komin00], Kominek, J., "Introduction to Fractal compression (long)", last referenced July 2000, http://www.faqs.org/faqs/compression-faq/part2/section-8.html.

[Krav99], Kravtchenko, V., "Tracking Color Objects in Real Time", M.Sc. Thesis, November 1999, http://www.cs.ubc.ca/grads/resources/thesis/Nov99/Vladimir-Kravtchenko.pdf.

[Krav(2)99], Kravtchenko, V., Little, J. J., "Efficient Color Object Segmentation Using the Diachromatic Reflection Model", IEEE PACRIM , 1999, ftp://ftp.cs.ubc.ca/pub/local/vk/pacrim.zip.

[Lawyer97], Lawyer, D. S., "Terminal Languages for Encoding Bit-Mapped Font", July 1997, http://www.funet.fi/pub/culture/russian/comp/cyril-term/font_langs.html.

[Lee84], Lee, B. G., "A New Algorithm to Compute the Discrete Cosine Transform", IEEE Transactions on Acoustics, Speech and Signal Processing, pp. 1243-1245, December 1984.

[Li00], Li, Z-N., Yang, Y., "Basics of Video", 2000, http://www.cs.sfu.ca/CourseCentral/365/li/material/notes/Chap3/Chap3.4/Chap3.4.html.

[Lohsc84}, Lohscheller, H., "Subjectively Adapted Image Communication System", IEEE Transactions on Communications, pp. 1316-1322, December 1994.

[Manning00], Manning, C. E., "Interframe Compression Techniques", last referenced May 2000, http://newmediarepublic.com/dvideo/compression/adv07.html.

[Marsh94], Marshall, D., "Introduction to Optical Flow", 1994-1997, http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/MARSHALL/node46.html.

[McKen99], McKenna, S. J., Raja, Y., Gong, S., "Tracking Colour Objects Using Adaptive Mixture Models", Image and Vision Computing, 17, pp. 225–231, 1999.

[Meier98], Meier, T., Ngan, K. N., "Automatic Segmentation of Moving Objects for Video Object Plane Generation", IEEE Transactions on Circuits and Systems for Video Technology, pp. 525-538, September 1998.

[Morse98], Morse, B. S., "Lecture 18: Segmentation (Region Based)", 1998-2000, http://www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/MORSE/region.pdf.

[MPEG-01], "MPEG Pointers and Resources", last referenced June 2001, http://www.mpeg.org/MPEG/mpeg.

[MPEG-4-01], "Overview of the MPEG-4 Standard", INTERNATIONAL ORGANISATION FOR STANDARDISATION, ISO/IEC JTC1/SC29/WG11, CODING OF MOVING PICTURES AND AUDIO", 2001, http://mpeg.telecomitalialab.com/standards/mpeg-4/mpeg-4.htm.

[Musman85], Musmann, H. G., Pirsch, P., Grallert, H-J., "Advances in Picture Coding", Proceedings of the IEEE, vol. 73, pp. 523-548, April 1985.

[Ohm97], Ohm, J-R., Ma, P., "Feature-Based Cluster Segmentation of Image Sequences", IEEE Int. Conf. on Image Processing, Vol. I, pp. 178-181, 1997, http://citeseer.nj.nec.com/349493.html.

[Osberg97], Osberger, W., Maeder, A., McLean, D., "A computational model of the human visual system for image quality assessment", Proceedings DICTA-97, pp. 337-342, 1997.

[Park99], Park, H. S., Ra, J., B., "Efficient image Segmentation Preserving Semantic Object Shapes", IEICE Transactions on Fundamentals, no. 6, pp. 879-886, June 1999.

[Penne93], Pennebaker, W. B., Mitchell, J. L., "JPEG Still Image Data Compression Standard", Van Nostrand Reinhold, 1st Edition, 1993.

[Picker97], Pickering, M. R., Arnold, J. F., Frater, M. R., "An Adaptive Search Length Algorithm for Block Matching Motion Estimation", IEEE Transactions on Circuits and Systems for Video Technology, pp. 906-912, December 1997.

[Phamdo00], Phamdo, N., "Vector Quantization", 2000-2001, http://www.data-compression.com/vq.html.

[Polikar99], Polikar, R., "The Wavelet Tutorial", 1999, http://www.public.iastate.edu/~rpolikar/WAVELETS/WTtutorial.html.

[Poynton96], Poynton, C., "A Technical Introduction to Digital Video", John Wiley & Sons, 1996.

[Sams97], Gurewich, O., Gurewich, N., "Teach Yourself Visual C++ 5 in 21 Days", Sams Publishing, 4th Edition, 1997.

[Shannon48], Shannon, C. E., "A Mathematical Theory of Communication", The Bell System Technical Journal, vol. 27, pp. 379-423, July 1948.

[Smith95], Smith, S. M., "ASSET-2: Real-Time Motion Segmentation and Object Tracking", Technical Report TR95SMS2b, British Crown Copyright 1995, http://www.fmrib.ox.ac.uk/~steve/asset/asset2/asset2.html.

[Smith97], Smith, S. M., Brady, J. M., "SUSAN – A New Approach to Low Level Image Processing", Int. Journal of Computer Vision, 23(1), pp. 45-78, May 1997.

[Sound01], Sound Vision Inc., "How Digital Cameras Work", May 2001, http://www.soundvisioninc.com/howdcw.htm.

[Sriniv85], Srinivasan, R., Rao, K. R., "Predictive Coding Based on Efficient Motion Estimation", IEEE Transactions on Communications, pp. 888-896, August 1985.

[Symes98], Symes, P., "Video Compression Demystified", McGraw-Hill, 1998.

[Taub86], Taub, H., Schilling, D. L., "Principles of Communication Systems", McGraw-Hill Company, 2nd Edition, 1986.

[Tele00], Telenor, last referenced March 2000, http://www.nta.no/brukere/DVC/.

[Teo94], Teo, P., Heeger, D., "Perceptual Image Distortion", Human Vision, Visual Processing and Digital display V, SPIE Proceedings, Vol 2179, pp. 127-141, 1994.

[Thomp98], Thompson, L., Wolf, P., "DVDO PureProgressive™ Digital Video Technology", 1998, http://www.projector-central.com/dvdo_pureprogressive.htm.

[Timmer99], Timmermans, M., "Bijective Arithmetic Encoding with Optimal End Treatment", September 1999, http://www3.sympatico.ca/mt0000/biacode/biacode.html.

[Tosun99], Tosun, A. S., "Video Compression: MPEG-4 and Beyond", November 1999, ftp://ftp.netlab.ohio-state.edu/pub/jain/courses/cis788-99/compression/index.html.

[Tour99], Tourapis, A. M., Au, O. C., Liou, M. L., Shen, G., "An Advanced Zonal Block Based Algorithm for Motion Estimation", IEEE International Conference on Image Processing (ICIP'99) Proceedings, section 26PO3.1, Kobe, Japan, October 1999, http://citeseer.nj.nec.com/tourapis99advanced.html.

[TriMedia], "TM1300 IREF Reference Manual", version 1.0, Momentum Data Systems.

[Turi96], Turi, R. H., Ray, S., "A new approach to clustering-based colour image segmentation", Proceedings of the IASTED International Conference. Signal and Image Processing (SIP-96), IASTED/ACTA Press, Anaheim USA, pp. 345-349, 1996.

[Uys00], Uys, R. F., "Parallel Implementation of Fractal Image Compression", M.Sc. Thesis, University of Natal, 2000.

[Vais92], Vaisey, J., Gersho, A., "Image Compression with Variable Block Size Segmentation", IEEE Transactions on Signal Processing, pp. 2040-2060, August 1992.

[Valens99], Valens, C., "A Really Friendly Guide to Wavelets", 1999, http://perso.wanadoo.fr/polyvalens/clemens/wavelets/wavelets.html.

[van Voor74], van Voorhis, D. C., "Constructing Codes with Bounded Codeword Lengths", IEEE Transactions on Information Theory, pp. 288-290, March 1974.

[Vide99], Video Development Initiative, "Digital Video for the Next Millennium", 1999, http://www.vide.net/resources/whitepapers/video/1.shtml.

[Weeks96], Weeks Jr., A. R., "Fundamentals of Electronic Image Processing", SPIE/IEEE Series on Imaging Science & Engineering, 1996.

[Welch84], Welch, T. A., "A Technique for High-Performance Data Compression", Computer, pp. 8-19, June 1984.

[Welch02], Welch, G., Bishop, G., "An Introduction to the Kalman Filter", UNC-Chapel Hill, TR 95-041, March 2002, http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf.

[Wiseman01], Wiseman, J., "An Introduction to MPEG Video Compression", last referenced March 2001, http://members.aol.com/symbandgrl/.

[Witt87], Witten, I. H., Neal, R. M, Cleary, J. G., "Arithmetic Coding for Data Compression", Communications of the ACM, pp. 520-540, June 1987.

[Ziv77], Ziv, J., Lempel, A., "A Universal Algorithm for Sequential Data Compression", IEEE Transactions on Information Theory, pp. 337-343, May 1977.

[Ziv78], Ziv, J., Lempel, A., "Compression of Individual Sequences via Variable-Rate Coding", IEEE Transactions on Information Theory, pp. 530-536, September 1978.