

BURRAQ – a Journey to NO Limits

(In collaboration with iPC, MCS)



By

Maj Shafqat Saeed

Capt Rai Sabir Hussain (Group Leader)

GC Umar Mahboob

Submitted to the Faculty of Computer Science Department
Military College of Signals, National University of Sciences and
Technology, Rawalpindi in partial fulfillment for the requirements of a B.E
Degree in Computer Software Engineering

JULY 2010

Abstract

Pakistan being a developing country does not possess the economy as well as state of the art technologies to produce modern and expensive defense equipment involving highly expensive research and development facilities. Therefore a large percentage of defense products are procured from developed nations like USA, France, Germany, UK etc and the procurement is subject to highly strict intellect and technical rights. Unmanned aerial vehicles are amongst the latest gadgets developed based on the modern technologies and hence are a trade mark only to the armies of developed modern countries. Unmanned Ariel Vehicles are very expensive equipment and the existing version of the BURRAQ (UAV) comes with inherent limitations.

The focus of the current research is to carry out study of the existing OEM specific media player (installed at BURRAQ UAV system), remove interlacing of cameras in the video caused due to proprietary implementation of the MPEG standard, and generate flicker less video at display, save flicker less videos of all or any number of cameras for later viewing and analysis, compress the generated video for further distribution through existing Communication mediums and standards, extract GPS data encrypted into the audio signal, and design as well as develop a customized enhanced version of the system.

DECLARATION

No portion of the work presented in this dissertation has been submitted in support of any other award or qualification either at National University of Sciences and Technology or any other institution.

DEDICATION

We dedicate our work to Pakistan.

ACKNOWLEDGMENTS

We are very grateful to Allah Almighty for giving us the confidence and strength for carrying out a research based project and for making us able to come up with an application that will be highly useful to the Pakistan military.

We also thank our instructors and friends for their moral and technical suggestions without which we might not have made it this far. We are thankful to Maj. Dr Naveed Iqbal Rao for guidance and help to complete this project successfully. Above all, we are deeply thankful to our beloved parents for their serenity, support and prayers that helped us gain our goal.

Contents

Abstract.....	ii
Introduction.....	4
1.1 Introduction.....	5
1.2 Scope and Objectives of the Project.....	5
1.3 Potential Difficulties and Problems.....	6
1.4 Intended Audience.....	7
Related Work.....	8
2.1 Existing System.....	9
2.1.1 Video Data.....	9
2.1.2 Audio Data.....	10
2.2 Problems in Existing System.....	10
2.3 Analysis of Existing System.....	11
2.3.1 Audio Analysis.....	12
2.3.2 Video Analysis.....	13
2.3.3 Software Analysis tools being used in research.....	14
2.4 Details of MPEG Standard used.....	18
2.4.1 MPEG-4 Standard.....	18
2.5 Techniques to hide data in audio.....	21
2.5.1 Low-bit coding.....	22
2.5.2 Phase Coding.....	22
2.5.3 Parity coding.....	23
2.5.4 Spread Spectrum.....	24
2.5.5 Echo Hiding.....	26
Requirement specifications.....	27
3.1 Functional Requirements of the Project.....	28
3.2 Non Functional Requirements of the Project.....	29
System Design.....	39
4.1 Design of Proposed System.....	40
4.1.1 Main Interface.....	40
4.1.2 Video Display.....	40

4.1.3	Video Generator	40
4.1.4	Video Compressor.....	41
4.1.5	Audio Data Reader.....	41
4.1.6	Frame Tagger	41
4.2	Proposed System Architecture.....	42
4.3	Use Case Diagram for BURRAQ.....	43
4.3.1	Use case UC1: AV Signal	44
4.3.2	Use case UC2: MPEG Standard.....	44
4.3.3	Use case UC3: GUI for Geographical Tagging	45
4.3.4	Use case UC4: Extended Interface	45
4.4	Sequence Diagrams.....	47
4.4.1	Sequence Diagram of UC1: AV Signal	47
4.4.2	Sequence Diagram of UC2: MPEG Standard.....	47
4.4.3	Sequence Diagram of UC3: GUI for Geographical tagging.....	48
4.4.4	Sequence Diagram of UC4: Extended Interface	49
	Implementation	50
5.1	Main Interface	51
5.1.1	Capture Video	51
5.1.2	Normalize Video	57
5.2	Video Display	58
5.3	Video Generator	60
5.4	Video Compressor	64
5.5	Audio Data Reader.....	65
5.6	Frame Tagger	68
	Results and Analysis	69
6.1	Mpeg standard.....	70
6.2	Video display.....	70
6.3	Audio data reader.....	70
6.4	Compression	71
6.5	Video Generator	71
6.6	Frame Extractor.....	72
	Conclusions and Future Work.....	73

Bibliography 76

Chapter 1

Introduction

1.1 Introduction

An unmanned aerial vehicle (UAV; also known as a remotely piloted vehicle or RPV, or Unmanned Aircraft System (UAS)) is an aircraft that flies without a human crew on board. Their largest uses are in operations involving reconnaissance and surveillance missions, surgical strikes and covert destruction missions. To distinguish UAVs from missiles, a UAV is defined as a reusable, uncrewed vehicle capable of controlled, sustained and level flight, powered by a jet or reciprocating engine. Therefore, cruise missiles are not considered UAVs, because, like many other guided missiles, the vehicle itself is a weapon that is not reused, even though it is also unmanned and in some cases remotely guided. The version of the BURRAQ UAV has some limitations. Hence, in this project detailed research and study of this existing system is carried with an aim to overcome the existing problems and provide a technically sound and up to date software application.

1.2 Scope and Objectives of the Project

This project has following objectives

- a) Study the design and architecture of existing system with a view to understand the working and organization of different modules, using available software and system analysis techniques like Disassemblers and Decompilers, and carry out static and dynamic analysis of the program execution.
- b) Design and develop an interface to extract the GPS data encoded in the audio stream, for any further use with other mapping applications.

- c) Design and develop an interface to view the received video flicker less and separately for all or any cameras as selected by the viewer.
- d) Design and develop an algorithm as well as interface for generating flicker less video of all or any cameras, as selected by the user. This involves modifying the existing implementation of MPEG standard, as used by the on board video processor for generating video signal.
- e) Design and develop an interface to compress the generated video using existing mpeg standards so that it can be viewed later using any free source Media player without any flicker.
- f) Design and develop an interface, to extract the Image/Images of any location in video, as selected by the user, for further use in Image Based Maps.

1.3 Potential Difficulties and Problems

The system has some difficulties which are discussed as below

- a) As the existing system is without the source codes for its software modules. Hence studying and analyzing a source less application is quite difficult and requires a large number of software fields to be studied and explored.
- b) Analysis of such modules is only possible through generation of respective assemblies which requires understanding structure, organization and architecture of multiple programming languages including assembly, C, C# and many other intermediate languages.

- c) Proprietary encryption algorithms come with inherent security and secrecy features thus making them almost unbreakable hence making the analysis and study extremely difficult and technically demanding.

1.4 Intended Audience

This document is intended for

- a) **Developers**: in order to be sure they are developing the right project that fulfills requirements provided in this document.
- b) **Users**: in order to get familiar with the idea of the project and suggest other features that would make it even more functional.
- c) **Documentation writers**: to know what features and in what way they have to explain. What security technologies are required, how the system will response in each user's action etc.
- d) **Advanced end users, end users/desktop and system administrators**: in order to know exactly what they have to expect from the system, right inputs and outputs and response in error situations.

Chapter 2

Related Work

2.1 Existing System

Pakistan is currently having BURRAQ UAV on its inventory which is a lightweight medium range reconnaissance and surveillance UAV system. It is being used for reconnaissance and surveillance operations. The UAV transmits its data to a ground station where it is viewed using customized hardware and software applications provided by the manufacturer. BURRAQ UAV system is utilizing a customized media player, which decodes GPS data from audio stream using a proprietary format. If this GPS data can be retrieved, it can be of great help and can be a foundation to many other applications. Therefore this project involves detailed study and analysis of the existing system being used to decode and display the audio and video information transmitted by BURRAQ UAV. The UAV sends data to its ground station in the form of AV signal containing video streams captured by the cameras and an audio signal.

2.1.1 Video Data

Video signal is generated by six cameras that are sending their video data simultaneously on a wireless link at 30 frames per second. One camera is main camera, and others are sub cameras. Out of 30 frames per second, 25 fps belongs to main camera and one frame each for every auxiliary camera. During video broadcast, smooth transmission is not possible due to inherent structure of **MPEG standard** being used by the on board video processor and lot of flickering as well as loss of data is experienced.

Main Camera	Main Camera	Main Camera	Main Camera	Main Camera	Sub Cam 1	Main Camera	Main Camera	Main Camera	Main Camera
Main Camera	Sub Cam 2	Main Camera	Main Camera	Main Camera	Main Camera	Main Camera	Sub Cam 3	Main Camera	Main Camera
Main Camera	Main Camera	Main Camera	Sub Cam 4	Main Camera	Main Camera	Main Camera	Main Camera	Main Camera	Sub Cam 5

Frame Details of the existing video format

2.1.2 Audio Data

The audio signal is generated by encrypting the GPS data being calculated, through an onboard microcomputer. This audio signal is only readable by the BURRAQ MEDIA PLAYER.

2.2 Problems in Existing System

The existing system of BURRAQ has following limitations

- a) Audio signal containing the Geographical data of underlying video is readable only to the 'UAV Media Player' and remains hidden for any other module to work with.
- b) The GUI of the Ground Control System provides the geographical data only in 'read only mode' hence preventing any further automated use with other applications like Digital Maps, Google Maps etc.
- c) Existing GUI is virtually dumb providing no usable output for further use in area mapping and marking.

- d) The MPEG standard and its implementation used by the existing system provides a flickering video making it almost useless for any tactical analysis and planning.
- e) Existing system provides an interface only to work with videos of a specified area. Therefore Aerial photographs, that can be used to generate Image Based Maps, cannot be extracted from the video using available functionalities.
- f) Existing system does not provide any means for generating separate compressed videos for all cameras that can be further distributed to other HQs, where they can be viewed flicker less using any free source media player.

2.3 Analysis of Existing System

To improve/modify the functionality of an existing executable application, a detailed study of the design and architecture is required. In a normal scenario the source codes as well as the supporting documentation will provide required information about the system, but in international standard exchanges neither the source code nor the related documentation is provided with the equipment hence making it very difficult to understand the internal structures of a software system. For studying such systems however various tools and techniques have been developed and evolved in the form of applications that provide the machine code equivalent of any executable software. In multimedia related such applications following two major areas are focused

- a) Audio Analysis
- b) Video Analysis

2.3.1 Audio Analysis

Audio analysis refers to the extraction of information and meaning from audio signals for analysis, classification, storage, retrieval, synthesis, etc. Audio analysis is carried out to find out bit contents of any wave sound. The wave sound is converted to respective bits and then the pattern is observed and analyzed for any underlying information about the contents. In case of this project the audio generated through the UAV on board microcomputer consists of GPS data masked into audio through a proprietary algorithm. So the audio is analyzed both in frequency as well as time domain. Digital signal analysis was also carried out and bit packets were generated to find any patterns of GPS data but no favorable results could be found.

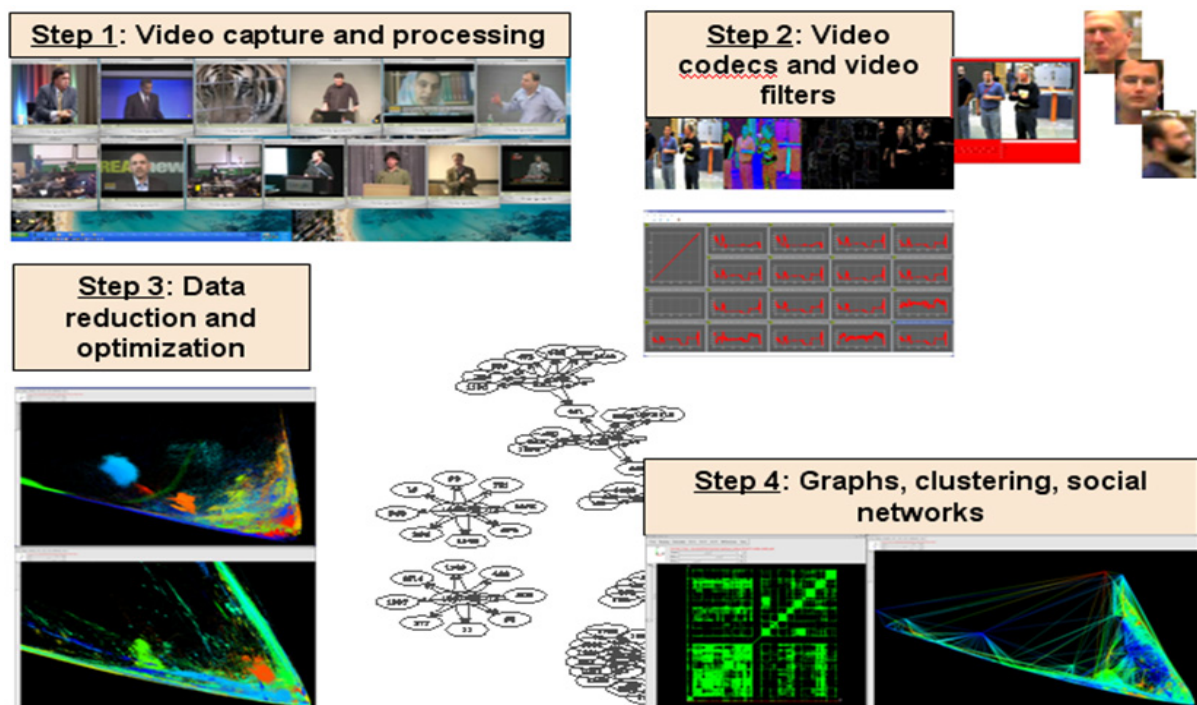


Audio analysis

2.3.2 Video Analysis

Video Content Analysis - is the capability of analyzing video to detect and determine temporal events not based on a single image. It is used in a wide range of domains including entertainment, health care, retail, automotive, transport, safety and security. The algorithms can be implemented as software on general purpose machines, or as hardware in specialized video processing units.

Video analysis is carried out to understand the placement and resemblance of frames in a particular video. The video is actually series of pictures bound together and displayed in a pattern depending upon the frame rate etc. Hence in video analysis, the video is broken up into individual frames to find the placement, pattern and rate of frames in a particular video.



Video Analysis

2.3.3 Software Analysis tools being used in research

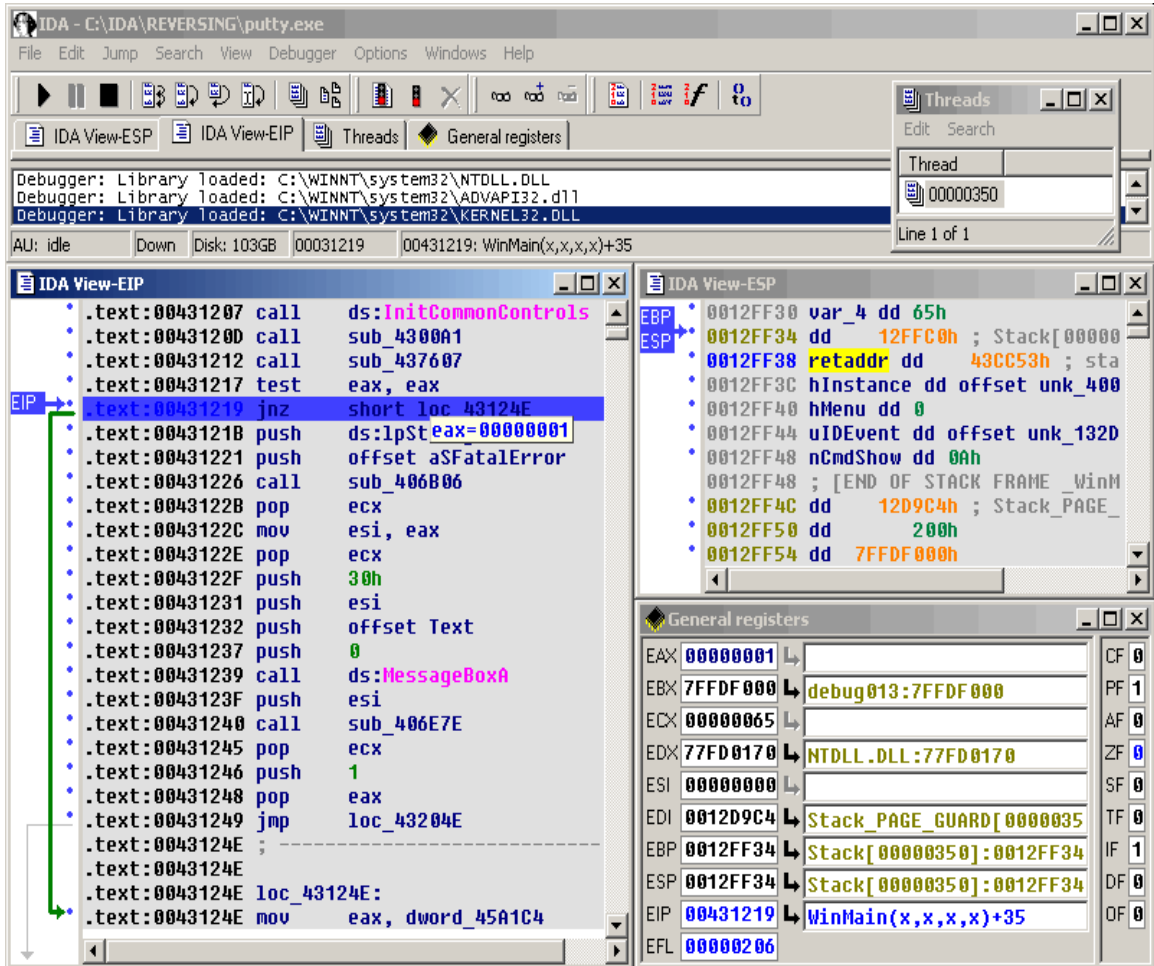
2.3.3.1 Disassembler

A disassembler is a computer program that translates machine language into assembly language—the inverse operation to that of an assembler. A disassembler differs from a decompiler, which targets a high-level language rather than an assembly language. Disassembly, the output of a disassembler, is often formatted for human-readability rather than suitability for input to an assembler, making it principally a reverse-engineering tool.

Assembly language source code generally permits the use of constants and programmer comments. These are usually removed from the assembled machine code by the assembler. If so, a disassembler operating on the machine code would produce disassembly lacking these constants and comments; the disassembled output becomes more difficult for a human to interpret than the original annotated source code. Some disassemblers make use of the symbolic debugging information present in object files such as ELF. The Interactive Disassemblers allow the human user to make up mnemonic symbols for values or regions of code in an interactive session: human insight applied to the disassembly process often parallels human creativity in the code writing process.

Disassembly is not an exact science: On CISC platforms with variable-width instructions, or in the presence of self-modifying code, it is possible for a single program to have two or more reasonable disassemblies. Determining which

instructions would actually be encountered during a run of the program reduces to the proven-unsolvable halting problem.



IDA Pro Disassembler

2.3.3.2 Decompilers

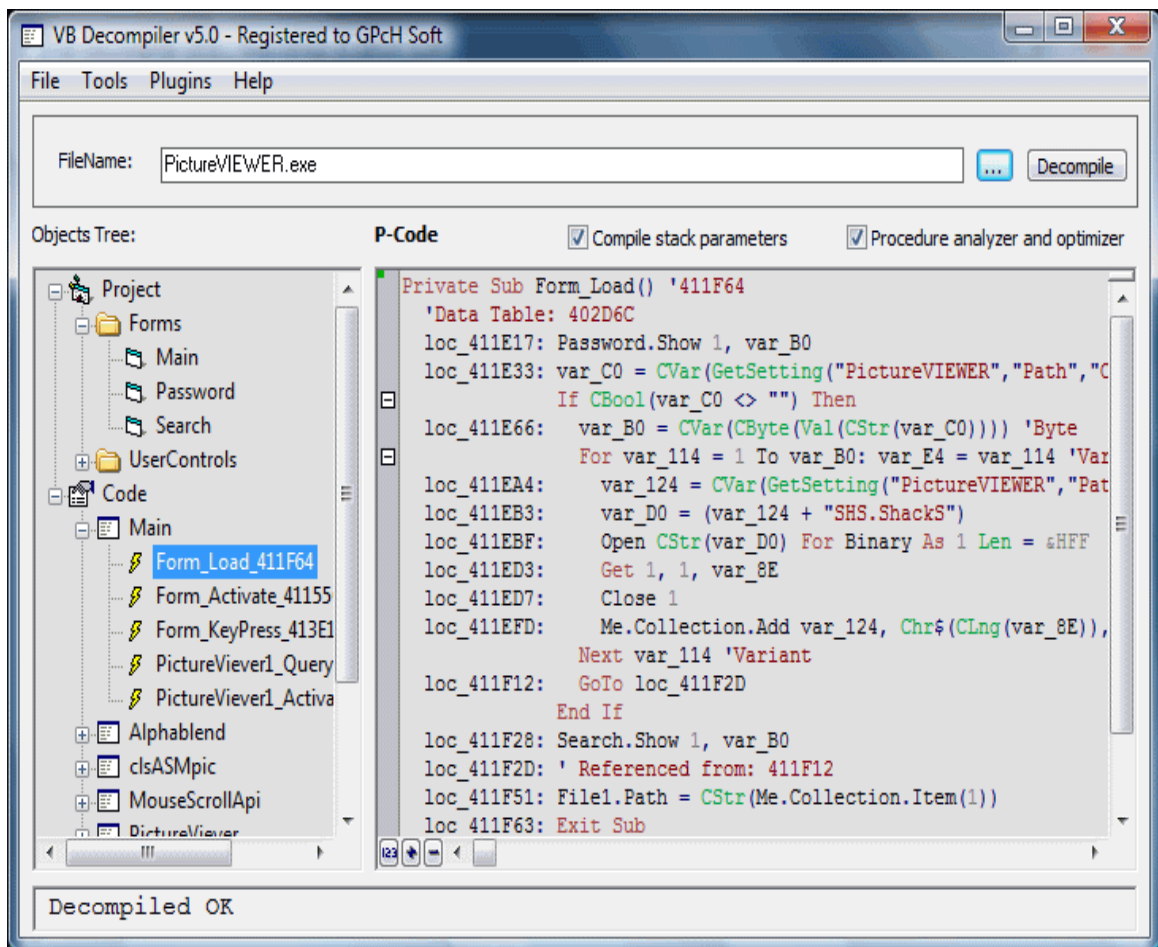
A decompiler is the name given to a computer program that performs the reverse operation to that of a compiler. That is, it translates a file containing information at a relatively low level of abstraction (usually designed to be computer readable rather than human readable) into a form having a higher level of abstraction (usually designed to be human readable).

The term *decompiler* is most commonly applied to a program which translates executable programs (the output from a compiler) into source code in a (relatively) high level language which, when compiled, will produce an executable whose behavior is the same as the original executable program. By comparison, a disassembler translates an executable program into assembly language (and an assembler could be used to assemble it back into an executable program).

Decompilation is the act of using a decompiler, although the term, when used as a noun, can also refer to the output of a decompiler. It can be used for the recovery of lost source code, and is also useful in some cases for computer security, interoperability and error correction. The success of decompilation depends on the amount of information present in the code being decompiled and the sophistication of the analysis performed on it. The bytecode formats used by many virtual machines (such as the Java Virtual Machine or the .NET Framework Common Language Run time) often include extensive metadata and high-level features that make decompilation quite feasible. The presence of debug data can make it possible to reproduce the original variable and structure names and even

the line numbers. Machine language without such metadata or debug data is much harder to decompile.

Some compilers and post-compilation tools produce obfuscated code (that is, they attempt to produce output that is very difficult to decompile). This is done to make it more difficult to reverse engineer the executable.



VB Decompiler

2.4 Details of MPEG Standard used

The **Motion Picture Experts Group (MPEG)** is a working group of experts that was formed by the ISO to set standards for audio and video compression and transmission. It is a group of standards for encoding and compressing audiovisual information such as movies, video, and music. MPEG is asymmetric in nature, as the compression process is time consuming and processor-intensive, whereas the decompression process is rapid and involves relatively inexpensive equipment. MPEG compression is as high as 200:1 for low-motion video of VHS quality, and broadcast quality can be achieved at 6 Mbps. Audio is supported at rates from 32 kbps to 384 kbps for up to two stereo channels. MPEG specifies lossy compression in the form of discrete cosine transform (DCT). MPEG is a joint technical committee of the International Standards Organization (ISO) and the International Electrotechnical Commission (IEC).

2.4.1 MPEG-4 Standard

MPEG-4 is a patented collection of methods defining compression of audio and visual (AV) digital data. It was introduced in late 1998 and designated a standard for a group of audio and video coding formats and related technology agreed upon by the ISO/IEC Moving Picture Experts Group (MPEG) (ISO/IEC JTC1/SC29/WG11) under the formal standard ISO/IEC 14496 - *Coding of audio-visual objects*. Uses of MPEG-4 include compression of AV data for web (streaming media) and CD distribution, voice (telephone, videophone) and broadcast television applications.

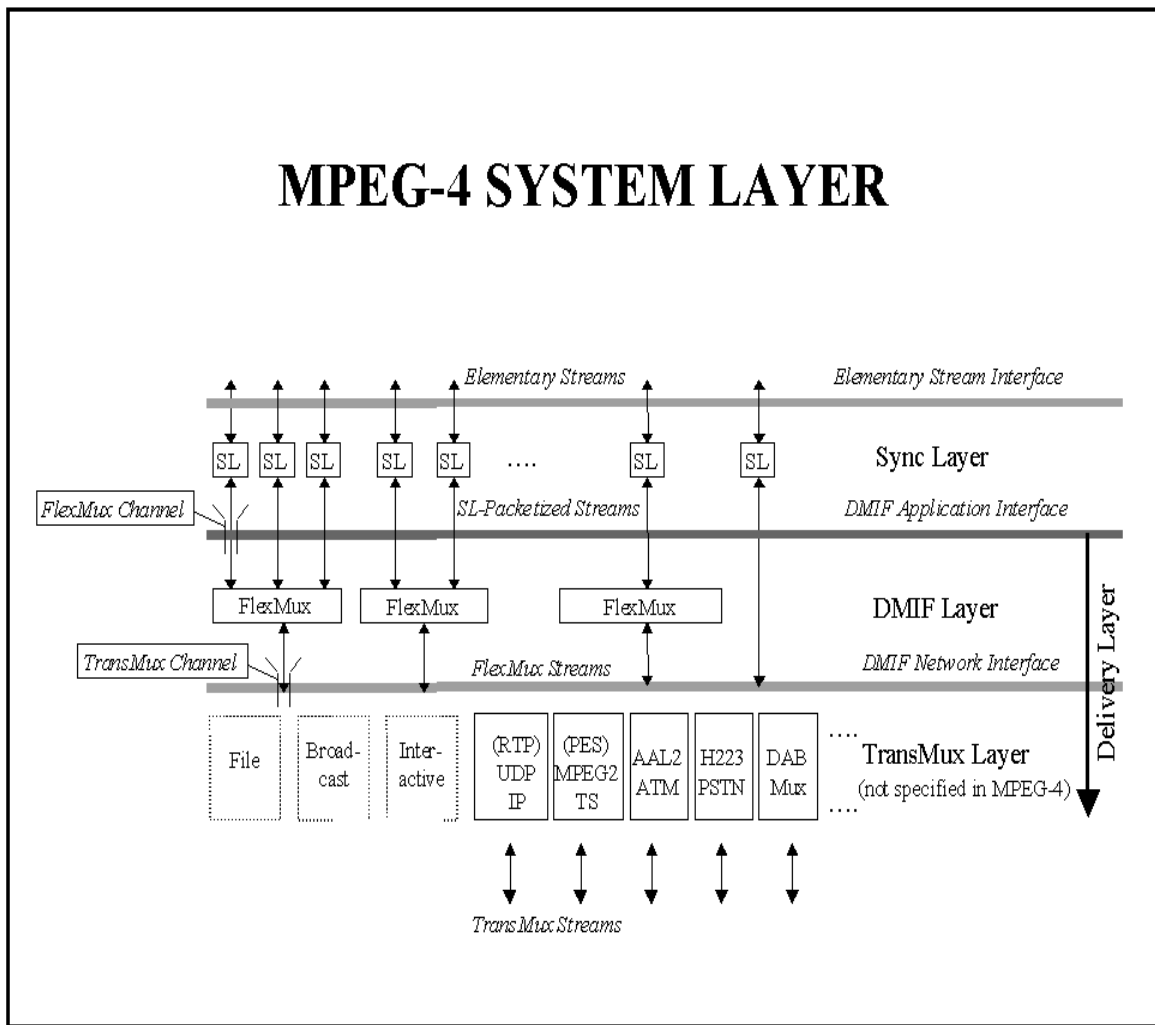
MPEG-4 absorbs many of the features of MPEG-1 and MPEG-2 and other related standards, adding new features such as (extended) VRML support for 3D rendering, object-oriented composite files (including audio, video and VRML objects), support for externally-specified Digital Rights Management and various types of interactivity. AAC (Advanced Audio Coding) was standardized as an adjunct to MPEG-2 (as Part 7) before MPEG-4 was issued.

Most of the features included in MPEG-4 are left to individual developers to decide whether to implement them. This means that there are probably no complete implementations of the entire MPEG-4 set of standards. To deal with this, the standard includes the concept of "profiles" and "levels", allowing a specific set of capabilities to be defined in a manner appropriate for a subset of applications.

Initially, MPEG-4 was aimed primarily at low bit-rate video communications; however, its scope as a multimedia coding standard was later expanded. MPEG-4 is efficient across a variety of bit-rates ranging from a few kilobits per second to tens of megabits per second. MPEG-4 provides the following functionalities:

- a) Improved coding efficiency over MPEG-2.
- b) Ability to encode mixed media data (video, audio, speech)
- c) Error resilience to enable robust transmission
- d) Ability to interact with the audio-visual scene generated at the receiver

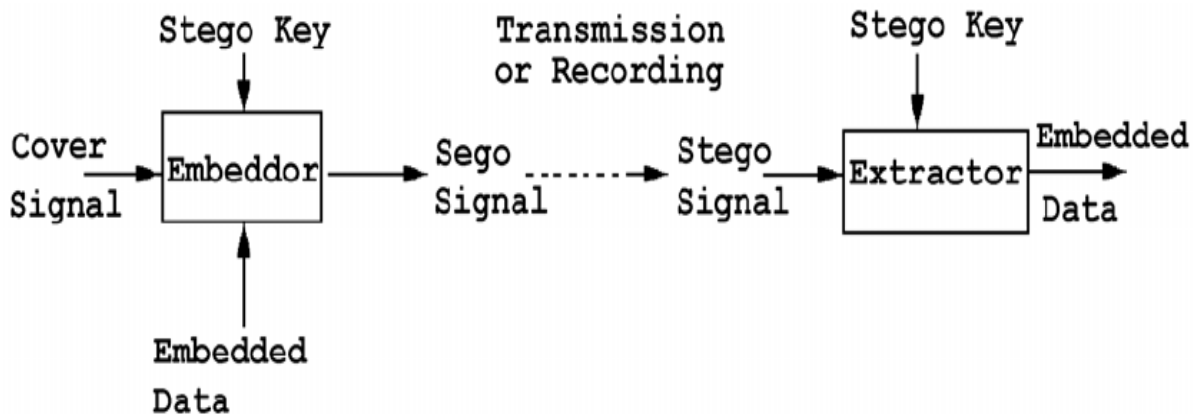
In the existing BURRAQ UAV system the implementation of this MPEG standard is used as such that it generates video of frames from different cameras. The placement of a different frame at every sixth index hence reduces the efficiency of the MPEG standard as well as introduces a flicker in the video that makes it jerky while viewing and analyzing.



MPEG-4 System Layer

2.5 Techniques to hide data in audio

Data hiding in audio signals is especially challenging, because the human auditory system (HAS) operates over a wide dynamic range. The HAS perceives over a range of power greater than one billion to one and a range of frequencies greater than one thousand to one. Sensitivity to additive random noise is also acute. The perturbations in a sound file can be detected as low as one part in ten million (80 dB below ambient level).



Block diagram of data hiding and retrieval.

However, there are some “holes” available. While the HAS has a large dynamic range, it has a fairly small differential range. As a result, loud sounds tend to mask out quiet sounds. Additionally, the HAS is unable to perceive absolute phase, only relative phase. Finally, there are some environmental distortions so common as to be ignored by the listener in most cases. We exploit many of these traits in the methods we discuss next, while being careful to bear in mind the extreme sensitivities of the HAS. A large number of techniques are used to hide data in audio. Some of them are explained below

2.5.1 Low-bit coding

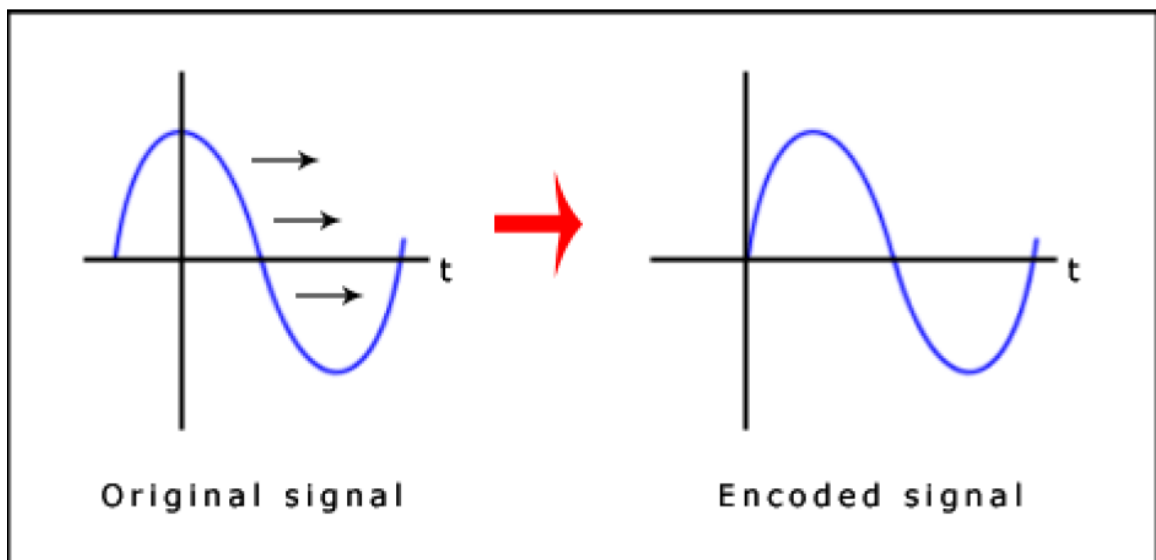
Low-bit coding is the simplest way to embed data into other data structures. By replacing the least significant bit of each sampling point by a coded binary string, we can encode a large amount of data in an audio signal.

Ideally, the channel capacity is 1 kb per second (kbps) per 1 kilohertz (kHz), e.g., in a noiseless channel, the bit rate will be 8 kbps in an 8 kHz sampled sequence and 44 kbps in a 44 kHz sampled sequence. In return for this large channel capacity, audible noise is introduced. The impact of this noise is a direct function of the content of the host signal, e.g., crowd noise during a live sports event would mask low-bit encoding noise that would be audible in a string quartet performance. Adaptive data attenuation has been used to compensate this variation.

2.5.2 Phase Coding

The phase coding method works by substituting the phase of an initial audio segment with a reference phase that represents the data. The phase of subsequent segments is adjusted in order to preserve the relative phase between segments. Phase coding, when it can be used, is one of the most effective coding methods in terms of the signal-to-perceived noise ratio. When the phase relation between each frequency component is dramatically changed, noticeable phase dispersion will occur.

However, as long as the modification of the phase is sufficiently small (sufficiently small depends on the observer; professionals in broadcast radio can detect modifications that are unperceivable to an average observer), an inaudible coding can be achieved. Phase coding relies on the fact that the phase components of sound are not as perceptible to the human ear as noise is. Rather than introducing perturbations, the technique encodes the message bits as phase shifts in the phase spectrum of a digital signal, achieving an inaudible encoding in terms of signal-to-perceived noise ratio.

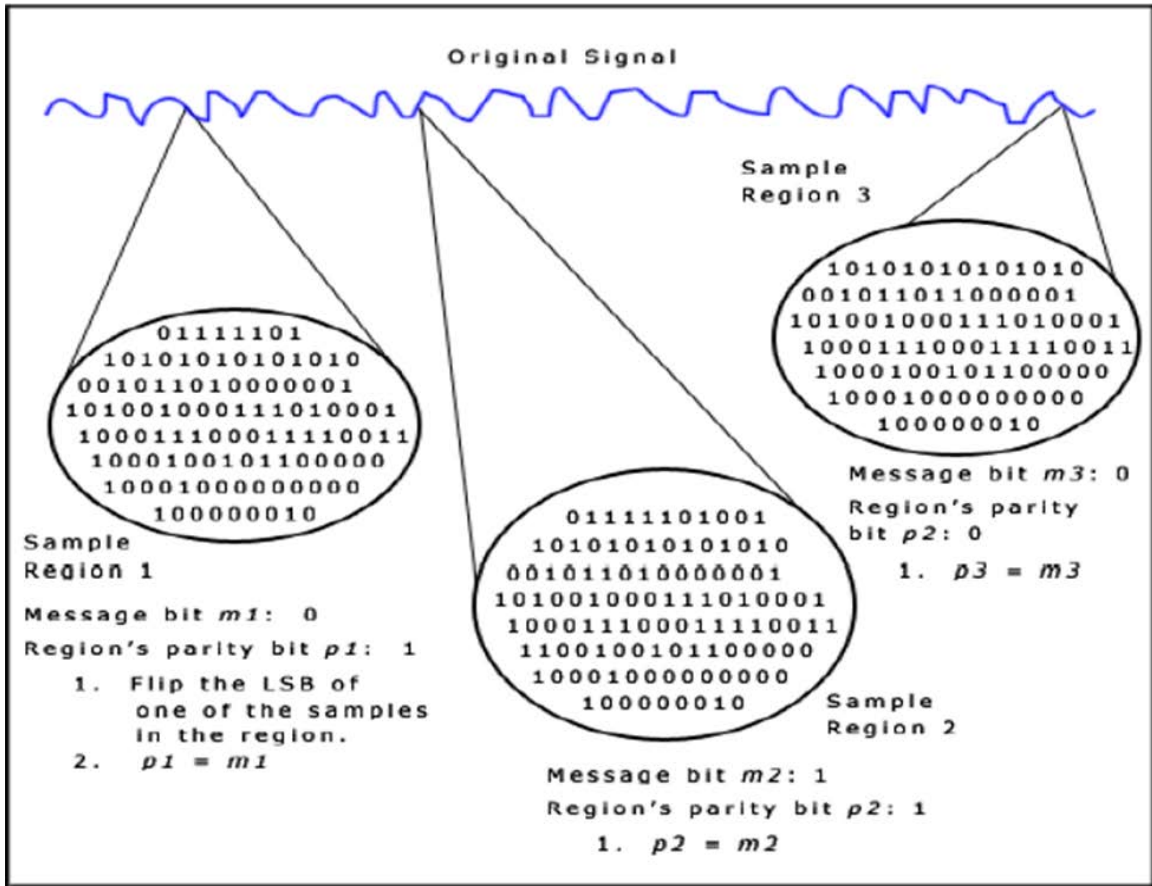


The signals before and after Phase coding procedure

2.5.3 Parity coding

One of the audio data hiding technique is parity coding technique. Instead of breaking a signal down into individual samples, the parity coding method breaks a signal down into separate regions of samples and encodes each bit from the secret message in a sample region's parity bit. If the parity bit of a selected region does not match the secret bit to be encoded, the process flips the LSB of

one of the samples in the region. Thus, the sender has more of a choice in encoding the secret bit, and the signal can be changed in a more unobtrusive fashion.

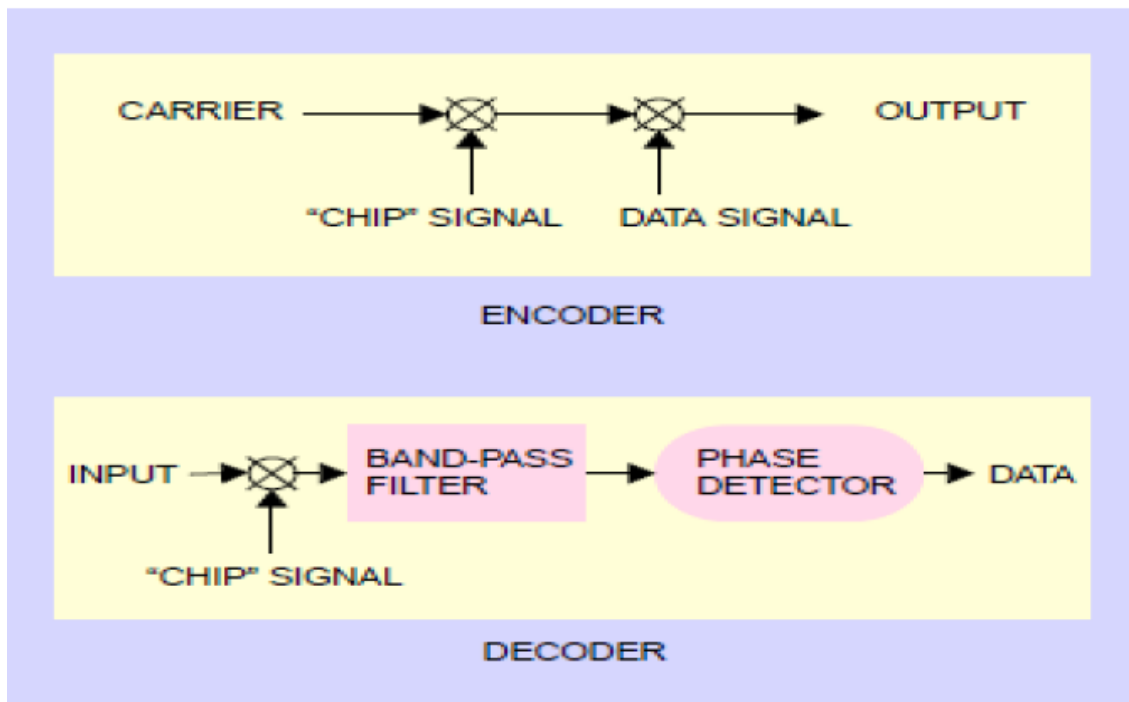


Parity Coding Procedure

2.5.4 Spread Spectrum

In a normal communication channel, it is often desirable to concentrate the information in as narrow a region of the frequency spectrum as possible in order to conserve available bandwidth and to reduce power. The basic spread spectrum technique, on the other hand, is designed to encode a stream of information by spreading the encoded data across as much of the frequency spectrum as possible. This allows the signal reception, even if there is

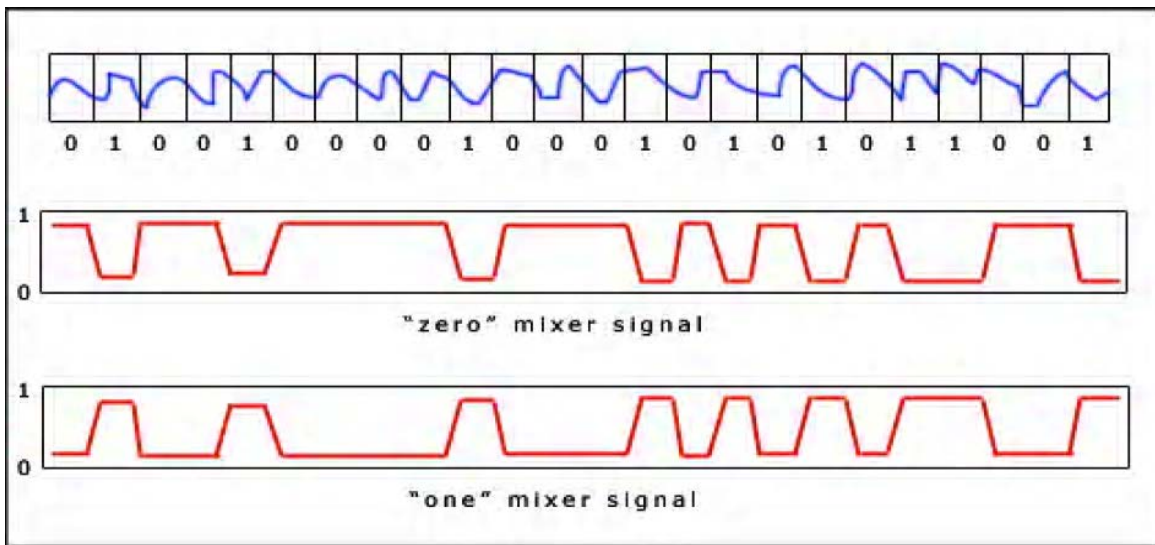
interference on some frequencies. While there are many variations on spread spectrum communication, we concentrated on Direct Sequence Spread Spectrum encoding (DSSS). The DSSS method spreads the signal by multiplying it by a chip, a maximal length pseudorandom sequence modulated at a known rate. Since the host signals are in discrete-time format, we can use the sampling rate as the chip rate for coding. The result is that the most difficult problem in DSSS receiving, that of establishing the correct start and end of the chip quanta for phase locking purposes, is taken care of by the discrete nature of the signal. Consequently, a much higher chip rate, and therefore a higher associated data rate, is possible. Without this, a variety of signal locking algorithms may be used, but these are computationally expensive.



Spread spectrum encoding

2.5.5 Echo Hiding

In echo hiding, information is embedded in a sound file by introducing an echo into the discrete signal. Like the spread spectrum method, it too provides advantages in that it allows for a high data transmission rate and provides superior robustness when compared to the noise inducing methods. If only one echo was produced from the original signal, only one bit of information could be encoded. Therefore, the original signal is broken down into blocks before the encoding process begins. Once the encoding process is completed, the blocks are concatenated back together to create the final signal.



Echo hiding

A message can also be encoded using musical tones with a substitution scheme. For example, a F# tone will represent a 0 and a C tone represents a 1. A normal musical piece can now be composed around the secret message or an existing piece can be selected together with an encoding scheme that will represent a message.

Chapter 3

Requirement

specifications

3.1 Functional Requirements of the Project

- a) The system should implement MPEG standard such that the flickers are removed and videos displayed separately for each camera selected.
- b) The system should be able to read the geographical data of ground locations (frames captured from onboard cameras) encoded in audio stream.
- c) The system should be able to generate separate compressed videos for all selected cameras using MPEG standard so that it can be viewed later without flicker and using any open source media player.
- d) The system should be able to tag the geographical info to its corresponding locations (Frames)
- e) The system should be able to retrieve locations (frames) based on geographical coordinates.(optional)
- f) The system should be able to provide compatible data for further use with applications like Google Maps etc.

3.2 Non Functional Requirements of the Project

a) **Security**

The system should be secure in a sense that the information should be received by the intended user only.

b) **Reliability**

The system should be reliable in a sense that the system should provide the users with the required functionality round the clock.

c) **Maintainability**

The system will be made maintainable so that in case of error or the user complaints the system might be changed to satisfy the new needs or to correct the errors.

d) **Reusability**

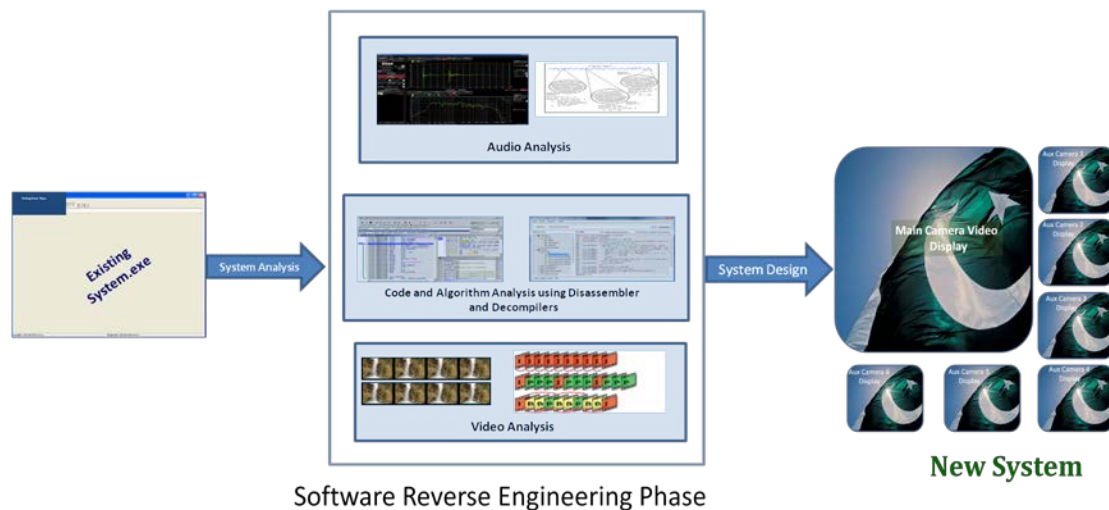
The system will be made reusable by making the application open source.

Chapter 4

Reverse Engineering **Process**

4.1 Reverse Engineering- definition

Reverse engineering means study and analyze an existing system in order to modify or reproduce the system. Software reverse engineering involves study of internal design and architecture to understand the existing software system and carry out required modifications.



4.2 Reverse engineering of Burraq

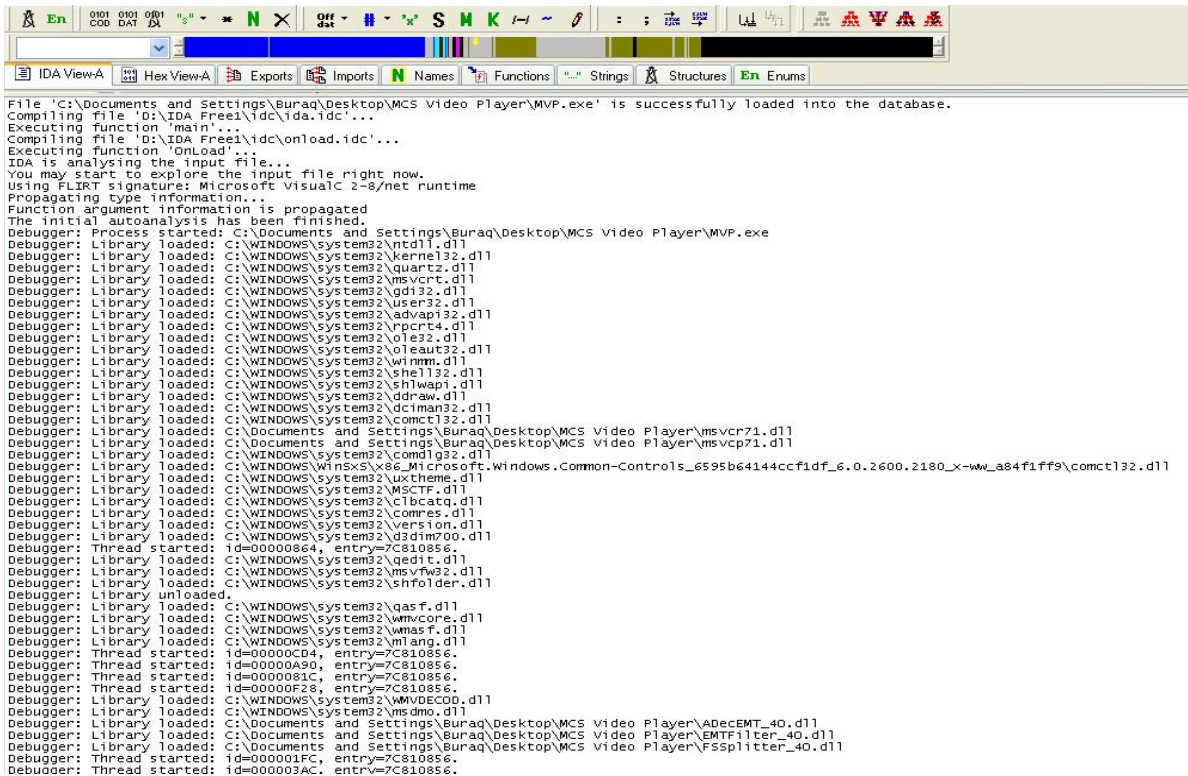
As discussed earlier the project involves reverse engineering of existing Burraq UAV system for:-

- Extraction of GPS data from Audio
- De Multiplex video to provide separate display

So disassembly of Burraq is done to generate assembly code of the exe application. This is done by using disassemblers that converts the machine code of an exe to its equivalent assembly codes. Programmers comments and constant strings are also generated back through some of the disassemblers. The complete process is describes in the following sections of this chapter.

4.2.1 Disassembly

The process of generating assembly code from the machine instructions generated by an exe or a program is called the disassembly process. The concerned application is loaded into disassembler along with its corresponding DLL's. a cold execution run is carried out and the machine calls traced. These machine calls or windows calls being used by the application are translated back to the assembly language.



```
File 'C:\Documents and Settings\Buraq\Desktop\MCS Video Player\MVP.exe' is successfully loaded into the database.
Compiling file 'D:\IDA Free1\idc\ida.idc'...
Executing function 'main'...
Compiling file 'D:\IDA Free1\idc\onload.idc'...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
Using FLIRT signature: Microsoft VisualC 2-8/net runtime
Propagating type information...
Function argument information is propagated
The initial autoanalysis has been finished.
Debugger: Process started: C:\Documents and Settings\Buraq\Desktop\MCS Video Player\MVP.exe
Debugger: Library loaded: C:\WINDOWS\system32\ntdll.dll
Debugger: Library loaded: C:\WINDOWS\system32\kernel32.dll
Debugger: Library loaded: C:\WINDOWS\system32\user32.dll
Debugger: Library loaded: C:\WINDOWS\system32\GDI32.dll
Debugger: Library loaded: C:\WINDOWS\system32\ole32.dll
Debugger: Library loaded: C:\WINDOWS\system32\oleaut32.dll
Debugger: Library loaded: C:\WINDOWS\system32\RPCRT4.dll
Debugger: Library loaded: C:\WINDOWS\system32\USER32.dll
Debugger: Library loaded: C:\WINDOWS\system32\ADVAPI32.dll
Debugger: Library loaded: C:\WINDOWS\system32\RPCRT4.dll
Debugger: Library loaded: C:\WINDOWS\system32\OLE32.dll
Debugger: Library loaded: C:\WINDOWS\system32\OLEAUT32.dll
Debugger: Library loaded: C:\WINDOWS\system32\WINMM.dll
Debugger: Library loaded: C:\WINDOWS\system32\SHELL32.dll
Debugger: Library loaded: C:\WINDOWS\system32\SHLWAPI.dll
Debugger: Library loaded: C:\WINDOWS\system32\DRAWAPI.dll
Debugger: Library loaded: C:\WINDOWS\system32\DCIMAN32.dll
Debugger: Library loaded: C:\WINDOWS\system32\COMCTL32.dll
Debugger: Library loaded: C:\Documents and Settings\Buraq\Desktop\MCS Video Player\msvcr71.dll
Debugger: Library loaded: C:\Documents and Settings\Buraq\Desktop\MCS Video Player\msvcp71.dll
Debugger: Library loaded: C:\WINDOWS\system32\COMD1G32.dll
Debugger: Library loaded: C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9\comctl32.dll
Debugger: Library loaded: C:\WINDOWS\system32\MSCTF.dll
Debugger: Library loaded: C:\WINDOWS\system32\CLBCATQ.dll
Debugger: Library loaded: C:\WINDOWS\system32\COMRES.dll
Debugger: Library loaded: C:\WINDOWS\system32\VERSION.dll
Debugger: Library loaded: C:\WINDOWS\system32\DDRAW.dll
Debugger: Library loaded: C:\WINDOWS\system32\GDI32.dll
Debugger: Thread started: 1d=00000864, entry=7C810856.
Debugger: Library loaded: C:\WINDOWS\system32\GDI32.dll
Debugger: Library loaded: C:\WINDOWS\system32\USER32.dll
Debugger: Library loaded: C:\WINDOWS\system32\SHLWAPI.dll
Debugger: Library loaded: C:\WINDOWS\system32\DRAWAPI.dll
Debugger: Library unloaded.
Debugger: Library loaded: C:\WINDOWS\system32\GAS.F.dll
Debugger: Library loaded: C:\WINDOWS\system32\WMVCORE.dll
Debugger: Library loaded: C:\WINDOWS\system32\WMASF.dll
Debugger: Library loaded: C:\WINDOWS\system32\MLANG.dll
Debugger: Thread started: 1d=00000CD4, entry=7C810856.
Debugger: Thread started: 1d=00000A90, entry=7C810856.
Debugger: Thread started: 1d=0000081C, entry=7C810856.
Debugger: Thread started: 1d=00000F26, entry=7C810856.
Debugger: Library loaded: C:\WINDOWS\system32\WVDECOD.dll
Debugger: Library loaded: C:\WINDOWS\system32\MSDMO.dll
Debugger: Library loaded: C:\Documents and Settings\Buraq\Desktop\MCS Video Player\ADecEMT_40.dll
Debugger: Library loaded: C:\Documents and Settings\Buraq\Desktop\MCS Video Player\EMTF11ter_40.dll
Debugger: Library loaded: C:\Documents and Settings\Buraq\Desktop\MCS Video Player\FSSP11ter_40.dll
Debugger: Thread started: 1d=000001FC, entry=7C810856.
Debugger: Thread started: 1d=000003AC, entry=7C810856.
```

Burraq's Disassembly Log

4.2.2 Disassembly Info

the disassembly process generates a lot of information about the software application like

- functions
- structures
- strings
- imports
- assembly codes
- names
- modules

The screenshot displays the IDA Pro interface with several windows open, illustrating the disassembly process. The 'Functions window' shows a list of subroutines with their segments and start addresses. The 'Structures' window displays a list of structures, including 'Ins/Del', 'D/R/*', 'N', 'U', and several collapsed structures. The 'Names window' lists various symbols and their addresses. The 'Strings window' shows a list of strings with their addresses, lengths, and types. The 'Imports' window lists imported functions from various libraries, such as 'RegCreateKeyA' and 'RegCloseKey'. The 'IDA View-A' window shows the disassembled code, including metadata like 'File Name', 'Format', 'Imagebase', and 'Section 1', as well as assembly code for a 'unicode' macro.

Disassembly Info

4.2.3 GPS Data Trace

The trace of the GPS data is made using the constants being used by the programmer like variable names Latitude, Longitude etc appearing on to the output. This trace involves physical inspection of the code to find the instances of the variables in the assembly code. Various features of the disassembler software helped in this analysis.

```

.rdata:0040FB50 ; DATA XREF: sub_404B90+667f0
.rdata:0040FB6A align 4
.rdata:0040FB6C ; char aType2DvFilesNo[]
.rdata:0040FB6C aType2DvFilesNo db 'Type2 DV files not supported.',0Ah,0
.rdata:0040FB6C ; DATA XREF: sub_404B90+649f0
.rdata:0040FB88 align 4
.rdata:0040FB8C ; char a20s[]
.rdata:0040FB8C a20s db '%20s',0 ; DATA XREF: sub_404B90+60Cf0
.rdata:0040FB91 align 4
.rdata:0040FB94 ; char a3_1fX3_1f[]
.rdata:0040FB94 a3_1fX3_1f db '(%3.1f! x %3.1f!)',0Ah,0 ; DATA XREF: sub_404B90+5F6f0
.rdata:0040FBA7 align 4
.rdata:0040FBA8 ; char aCamera11s[]
.rdata:0040FBA8 aCamera11s db 'Camera: %11s',0Ah,0 ; DATA XREF: sub_404B90+5CDf0
.rdata:0040FBB6 align 4
.rdata:0040FBB8 ; char aAltitudeAsl5_0[]
.rdata:0040FBB8 aAltitudeAsl5_0 db 'Altitude ASL:%5.01f%m',0Ah,0
.rdata:0040FBB8 ; DATA XREF: sub_404B90+5BCf0
.rdata:0040FBCF align 10h
.rdata:0040FBD0 ; char aAltitudeAgl_hg[]
.rdata:0040FBD0 aAltitudeAgl_hg db 'Altitude AGL: .hgt missing',0Ah,0
.rdata:0040FBD0 ; DATA XREF: sub_404B90+59Ff0
.rdata:0040FBEE align 10h
.rdata:0040FBF0 ; char aAltitudeAgl5_0[]
.rdata:0040FBF0 aAltitudeAgl5_0 db 'Altitude AGL:%5.01f%m',0Ah,0
.rdata:0040FBF0 ; DATA XREF: sub_404B90+58Cf0
.rdata:0040FC07 align 4
.rdata:0040FC08 ; char aHeading5_01fC[]
.rdata:0040FC08 aHeading5_01fC db 'Heading: %5.01f%c',0Ah,0
.rdata:0040FC08 ; DATA XREF: sub_404B90+562f0
.rdata:0040FC1F align 10h
.rdata:0040FC20 ; char aLongitude7_41f[]
.rdata:0040FC20 aLongitude7_41f db 'Longitude: %7.41f%c',0Ah,0 ; DATA XREF: sub_404B90+540f0
.rdata:0040FC35 align 4
.rdata:0040FC38 ; char aLatitude7_41fC[]
.rdata:0040FC38 aLatitude7_41fC db 'Latitude: %7.41f%c',0Ah,0 ; DATA XREF: sub_404B90+4F3f0
.rdata:0040FC38 ; sub_404B90+517f0

```

GPS data Trace

4.2.4 Variable Access flow charts

After tracing the instances of the variables, they are tracked for any read write operations being done using memory read write traces. Since the virtual addressing of the application remains same hence the memory locations can be traced using the virtual addresses. Using these read write traces a flow chart can be prepared depicting the program's complete working sequence till accessing our concerned variables. This flow chart gives us an abstract pictures of the assembly subroutines accessing our variables. This can be used in turn to trace the main functions dealing with read write operations. Example of one such flow chart is given in fig showing access trace for Latitude.

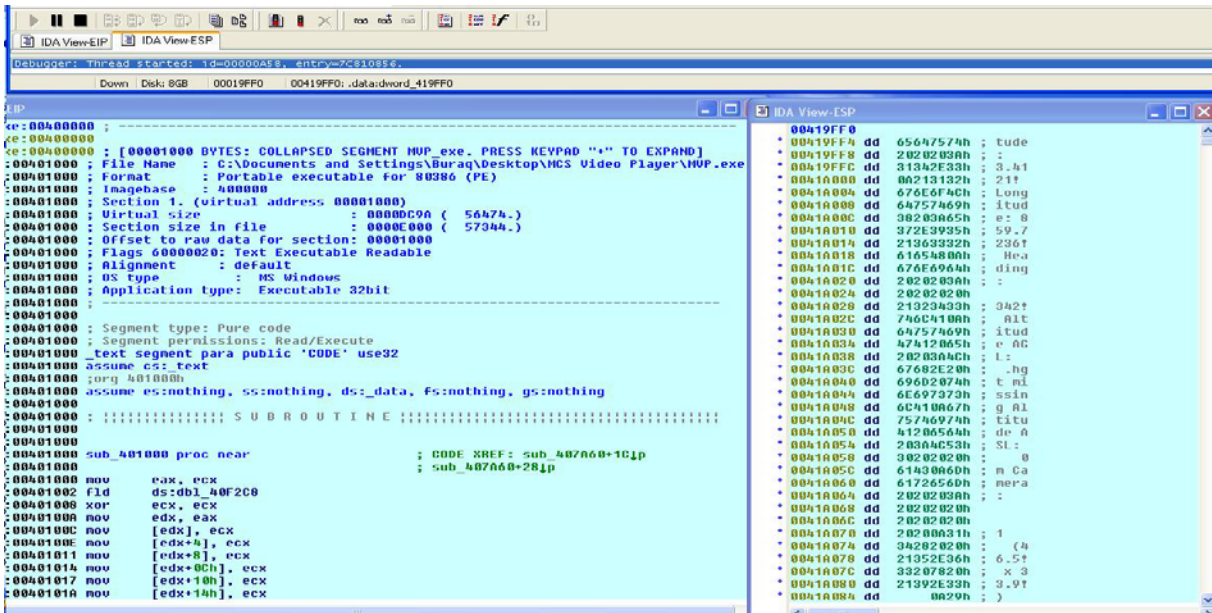
4.2.6 Memory Trace

After havinf traced all the functions and their relative argumernts, using the virtual addressing of the system, the key memory locations are traced both statically as well as dynamically. This analysis will confirm our initial findings about functions and their read write operations.

```
-:00419FF0 ; char dword_419FF0
-:00419FF0 dword_419FF0 dd 0
-:00419FF0 ; DATA XREF: sub_403940+5C70
-:00419FF0 ; sub_404B90+4D270 ...
-:00419FF4 db 0
-:00419FF5 db 0
-:00419FF6 db 0
-:00419FF7 db 0
-:00419FF8 db 0
-:00419FF9 db 0
-:00419FFA db 0
-:00419FFB db 0
-:00419FFC db 0
-:00419FFD db 0
-:00419FFE db 0
-:00419FFF db 0
-:0041A000 db ?
-:0041A001 db ?
-:0041A002 db ?
-:0041A003 db ?
-:0041A004 db ?
-:0041A005 db ?
-:0041A006 db ?
-:0041A007 db ?
-:0041A008 db ?
-:0041A009 db ?
-:0041A00A db ?
-:0041A00B db ?
-:0041A00C db ?
-:0041A00D db ?
-:0041A00E db ?
-:0041A00F db ?
-:0041A010 db ?
-:0041A011 db ?
-:0041A012 db ?
-:0041A013 db ?
-:0041A014 db ?
-:0041A015 db ?
-:0041A016 db ?
-:0041A017 db ?
-:0041A018 db ?
```

Static Memory Trace

Static memory trace provides us the information about a memory location before the execution of the program. Hence we get null values here at our concerned memory locations.



Dynamic Memory Trace

Unlike static memory trace, the dynamic trace yields the desired results by displaying the required GPS info at the concerned memory locations hence confirming our earlier findings of data.

4.2.7 Findings of Disassembly-Audio

The disassembly of Burraq yields following

- GPS data traced
- Virtual addresses known
- Uses windows API to Display decoded data
 - Msvcr71.dll
- Data accessible by hooking DLL communication
- Possible through injecting a code snippet

4.2.8 Video Data Trace

The disassembly also yields important information about the video data that was already acquired using the video analysis tech

```

.text:00404F64
.text:00404F64 loc_404F64:          ; DATA XREF: .text:00405258↓j
.text:00404F64      mov     ebx, offset aIr ; "IR"
.text:00404F69      jmp     short loc_404FE0
-----
.text:00404F6B
.text:00404F6B loc_404F6B:          ; CODE XREF: sub_404B90+3CD↑j
.text:00404F6B      ; DATA XREF: .text:0040525C↓j
.text:00404F6B      mov     ebx, offset aFront ; "Front"
.text:00404F70      jmp     short loc_404FE0
-----
.text:00404F72
.text:00404F72 loc_404F72:          ; CODE XREF: sub_404B90+3CD↑j
.text:00404F72      ; DATA XREF: .text:00405260↓j
.text:00404F72      mov     ebx, offset a1 ; "1"
.text:00404F77      jmp     short loc_404FE0
.text:00404F79 ; |
-----
.text:00404F79 loc_404F79:          ; CODE XREF: sub_404B90+3CD↑j
.text:00404F79      ; DATA XREF: .text:00405264↓j
.text:00404F79      mov     ebx, offset a2 ; "2"
.text:00404F7E      jmp     short loc_404FE0
-----
.text:00404F80
.text:00404F80 loc_404F80:          ; CODE XREF: sub_404B90+3CD↑j
.text:00404F80      ; DATA XREF: .text:00405268↓j
.text:00404F80      mov     ebx, offset a3 ; "3"
.text:00404F85      jmp     short loc_404FE0
-----
.text:00404F87
.text:00404F87 loc_404F87:          ; CODE XREF: sub_404B90+3CD↑j
.text:00404F87      ; DATA XREF: .text:0040526C↓j
.text:00404F87      mov     ebx, offset a4 ; "4"
.text:00404F8C      jmp     short loc_404FE0
-----
.text:00404F8E
.text:00404F8E loc_404F8E:          ; CODE XREF: sub_404B90+3CD↑j
.text:00404F8E      ; DATA XREF: .text:00405270↓j
.text:00404F8E      mov     ebx, offset aWing ; "Wing"
.text:00404F93      jmp     short loc_404FE0

```

Video Data Trace

The disassembly confirms our initial findings about the video data as

- Multiplexed video from input of seven cameras
- Single streams contains frames from all cameras
- 30 fps format
- 25 main camera frames and 5 Aux cam frames in each second

Chapter 5

System Design

5.1 Design of Proposed System

Keeping in view the requirement specifications of the system, following components are considered necessary to be designed and integrated

5.1.1 Main Interface

Main Interface would be the basic component of the system that receives the AV file containing audio and video data sent from the UAV. It would carry out the necessary normalizations of the file if required, generates the normalized video for further use by underlying components.

5.1.2 Video Display

This is the component that provides a smooth video at display by removing the flicker and setting the frames at respective display areas for different cameras. It would also provide the means of extracting any frame as required by the user and save to a desired location for further use.

5.1.3 Video Generator

This component would generate separate videos for respective cameras, overcoming the existing MPEG implementation weaknesses. The generated videos of selected cameras and their respective frames from the incoming video would be saved as separate video files at desired locations.

5.1.4 Video Compressor

This component would provide means to compress the generated videos using existing Mpeg standards so that later the videos can be viewed using any free source media players.

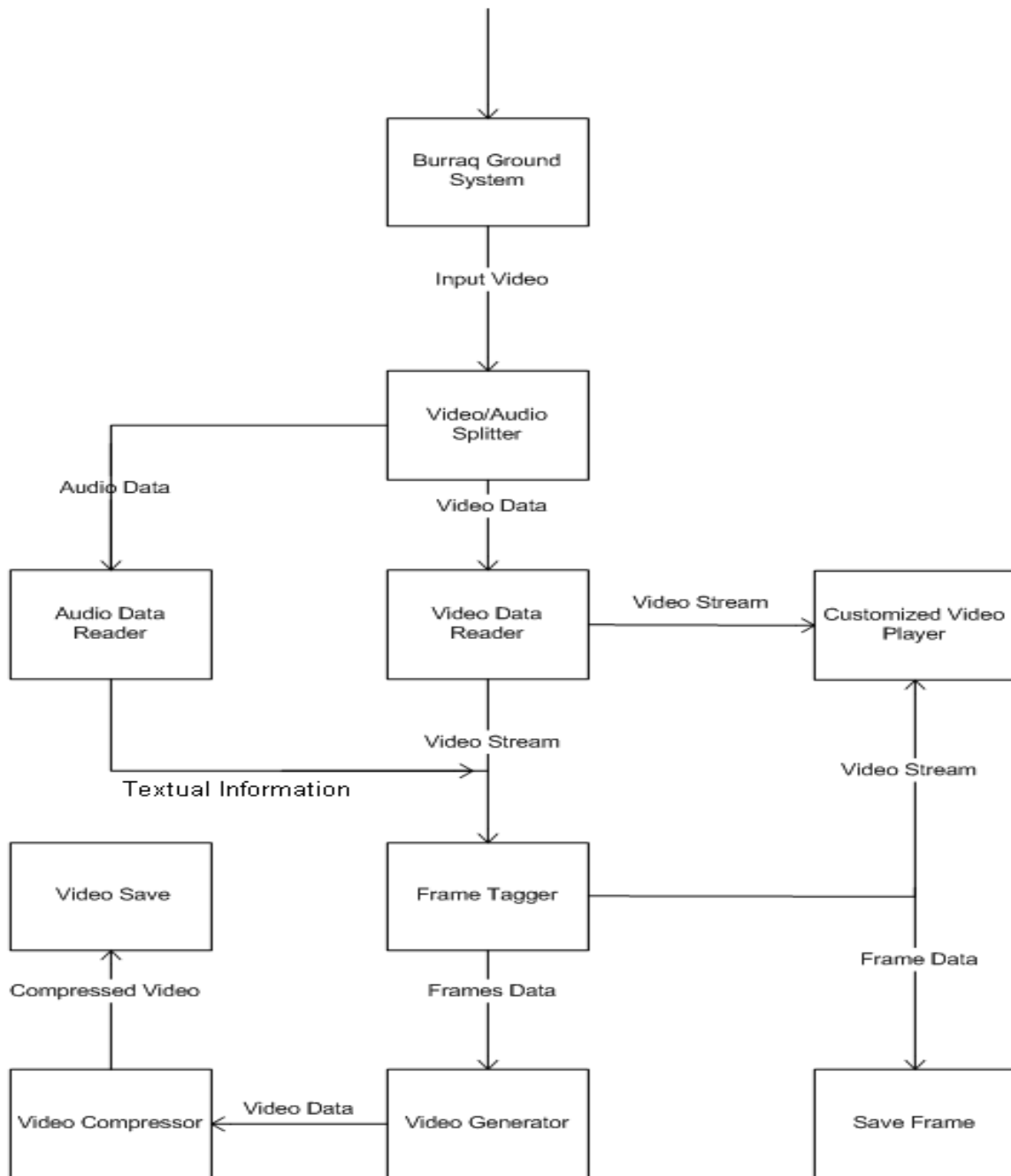
5.1.5 Audio Data Reader

It is a component that reads the data hidden in audio signal using existing media player and stores it to a desired location for further use. It is required either to decode the audio information using the encoding information or read the data directly from the memory of the existing media player.

5.1.6 Frame Tagger

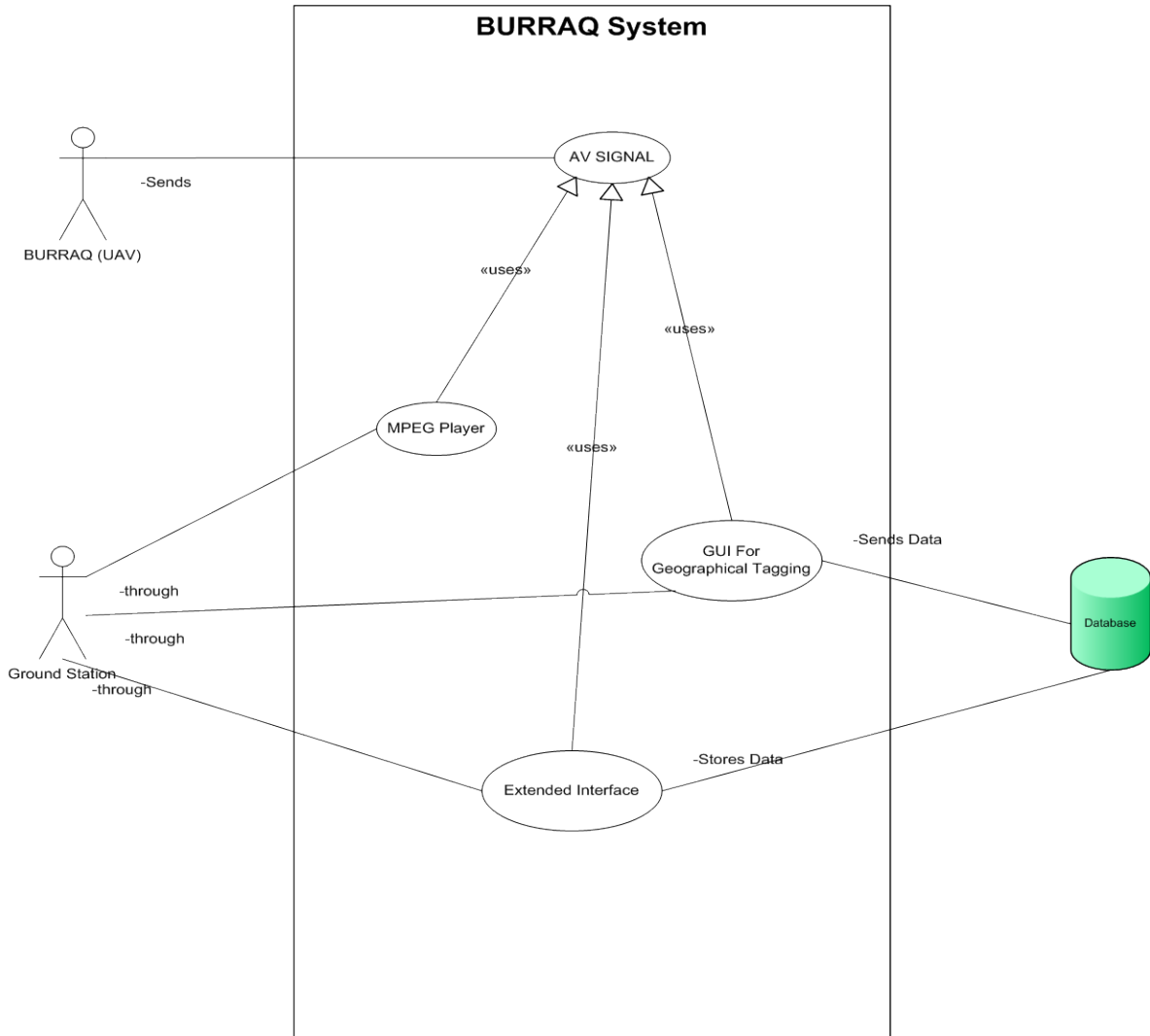
The component that, when required by the user, provides means to save any particular image of a location along with its GPS coordinates tagged, to a desired location.

5.2 Proposed System Architecture



Detailed Design Diagram

5.3 Use Case Diagram for BURRAQ



5.3.1 Use case UC1: AV Signal

- a) **Primary Actor:** UAV
- b) **Preconditions:** Connection with the Ground Station
- c) **Success Guarantee (Post conditions):** AV Signal is sent to the Ground Station
- d) **Main success scenario / Basic flow:**
 - 1. Ground Station gets the real time data in the form of AV Signal.
 - 2. Ground Station store the data in its archives.
- e) **Extensions/ Alternative flow:** N/A
- f) **Special requirements:** Video should be noise free and Camera should be calibrated.
- g) **Technology and Data Variations List:** Read video from camera if required.

5.3.2 Use case UC2: MPEG Standard

- a) **Primary Actor:** Ground Station
- b) **Preconditions:** AV Signal is available. Video is displayed with an inherent flicker.
- c) **Success Guarantee (Post conditions):** Outputs flicker less display of video
- d) **Main success scenario / Basic flow:**
 - 1. Ground Station uses the customized MPEG standard.
 - 2. Input to the MPEG Standard is the AV Signal obtained from the UAV.
 - 3. MPEG Standard removes the flickers from the video.
 - 4. MPEG Standard smoothes the displayed video.
 - 5. If required then the MPEG Standard stores the flicker less Video in the database.
- e) **Extensions/ Alternative flow:** N/A

- f) **Special requirements:** No special requirement
- g) **Technology and Data Variations List:** No technology and data variations.

5.3.3 Use case UC3: GUI for Geographical Tagging

- a) **Primary Actor:** Ground Station
- b) **Preconditions:** AV Signal is available.
- c) **Success Guarantee (Post conditions):**Extracts the geographical data (longitude and latitudes etc) from audio in the AV Signal of the path which has been traversed by BURRAQ.
- d) **Main success scenario / Basic flow:**
 - 1. Input to the interface is the AV signals.
 - 2. The interface extracts the geographical data from the audio.
 - 3. The system returns the output in the form of geographical data and stores the output to the database if needed.
- e) **Extensions/ Alternative flow:** N/A
- f) **Special requirements:** No special requirement
- g) **Technology and Data Variations List:** No technology and data variations

5.3.5 Use case UC4: Extended Interface

- a) **Primary Actor:** Ground Station
- b) **Preconditions:** Smoothened video is available in the database. Geographical coordinates are available in database. Frames are available.
- c) **Success Guarantee (Post conditions):** The interface tags the geographical coordinates onto the corresponding frames.

d) **Main success scenario / Basic flow:**

1. The ground station uses the extended GUI module.
2. Inputs to this module are maps and geographical coordinates which are obtained from the AV Signal by the Interface for geographical tagging.
3. Module then tags the map according to the coordinates.
4. Then it stores the output in the databases if needed.

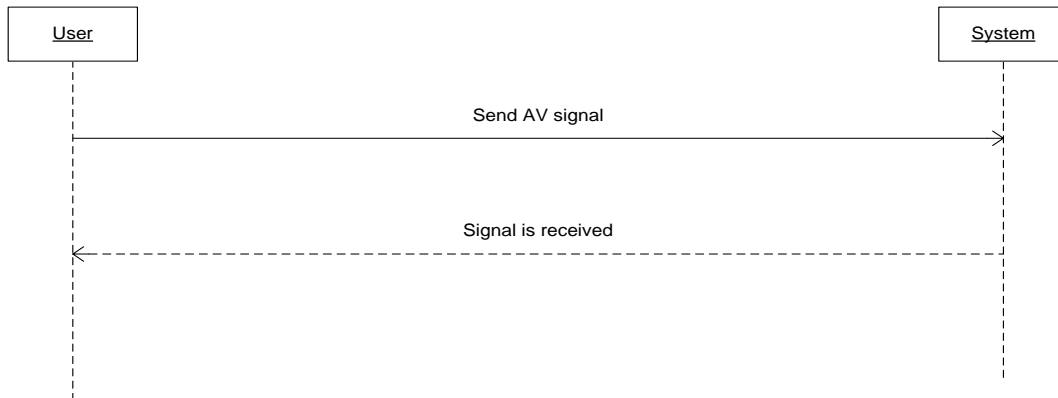
e) **Extensions/ Alternative flow:** N/A

f) **Special requirements:** No special requirement

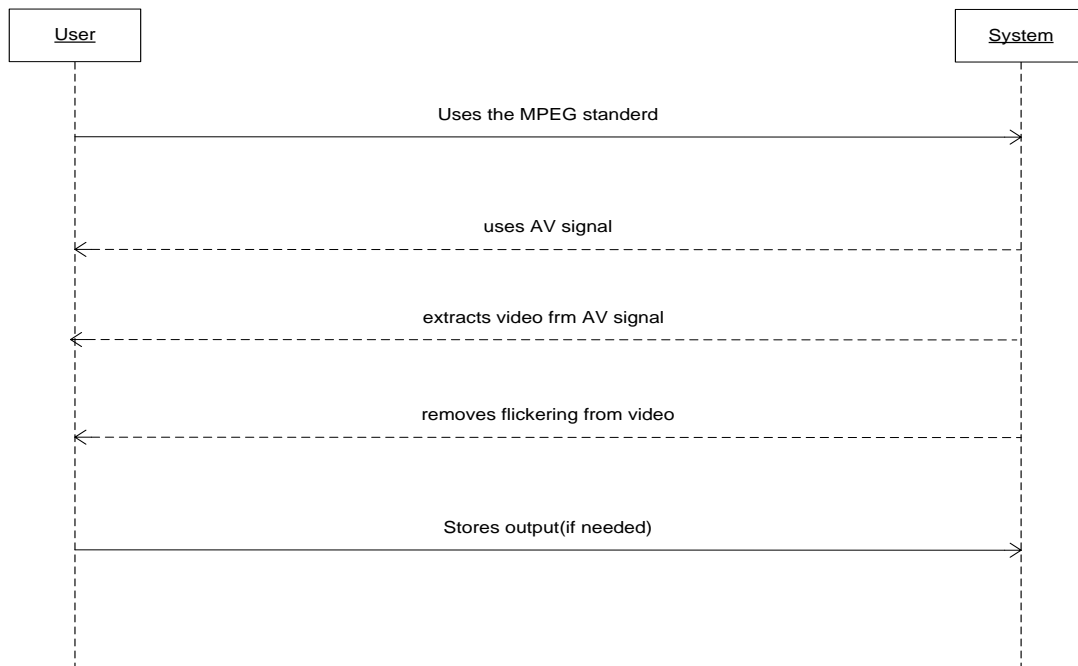
g) **Technology and Data Variations List:** No technology and data variations.

5.4 Sequence Diagrams

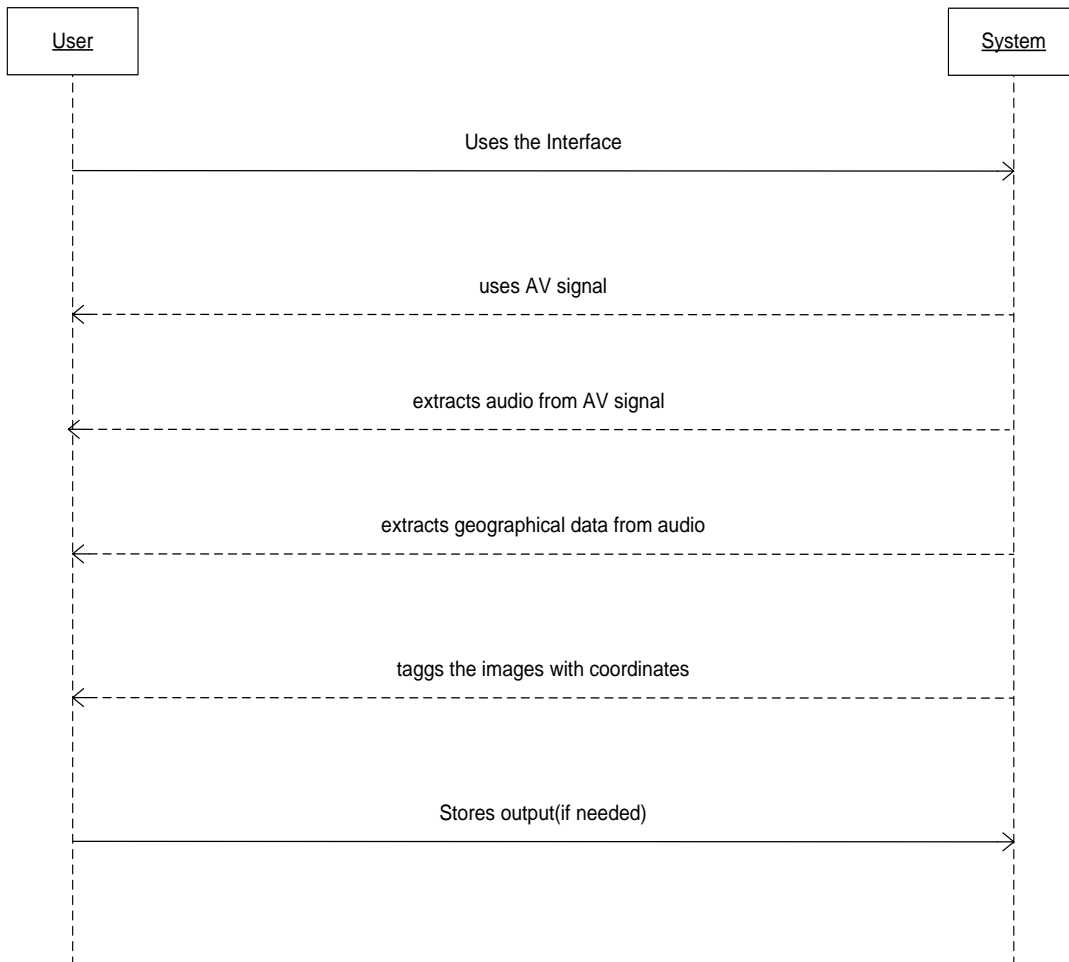
5.4.1 Sequence Diagram of UC1: AV Signal



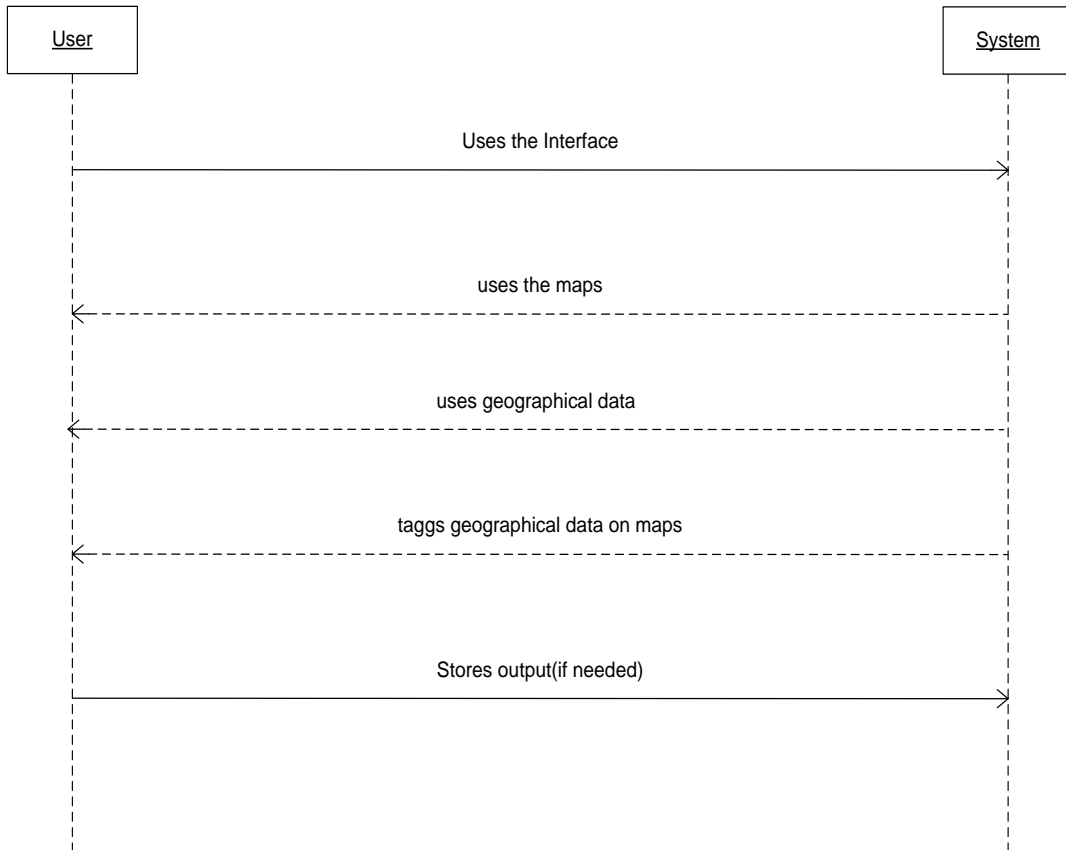
5.4.2 Sequence Diagram of UC2: MPEG Standard



5.4.3 Sequence Diagram of UC3: GUI for Geographical tagging



5.4.4 Sequence Diagram of UC4: Extended Interface



Chapter 6

Implementation

The designed system is implemented using the following methodologies discussed separately for each component

6.1 Main Interface

As discussed in design, the main interface is required to receive the AV file being sent down from UAV and normalize it if required. Hence the implementation of this component is divided into two parts

6.1.1 Capture Video

The basic requirement of the main interface is to capture the video into our system so that individual frames can be accessed later on when required. This feature is implemented using Microsoft® DirectShow® Editing Services (DES). It is an application programming interface (API) that greatly simplifies the tasks involved in video editing. DES is built on top of the core DirectShow architecture. It abstracts much of the complexity of DirectShow, and provides a set of interfaces designed specifically for manipulating video editing projects. As an application developer, you get the benefits of DirectShow inside a framework much better suited for creating video editing applications.

Using Direct Show lib's public interface IMediaDet we have created our own class that uses the functions of this API to actually capture the incoming video for further editing and use. Information from the incoming video file header is extracted to calculate Frame rate and length of the media. This information is

further used to calculate total number of frames in the video file. Hence it makes our input video file accessible at each frame separately.

GetImage (image number) is the implemented functions that provide us an interface to access each frame separately in the video. Our implemented class is compiled as a DLL file providing us all functionalities through an external interface.

Hence after the implementation of this component any incoming video can be captured and accessed at individual frame level. This implementation is shown through following code

```
namespace Burraq
{
    /// <summary>
    ///     Retrieves individual frames from a video file and returns them
    ///     as bitmaps.
    /// </summary>
    public class FrameGrabber : IEnumerable<FrameGrabber.Frame>
    {
        // underlying variables for properties
        private string fileName;
        private int frameCount;
        private double frameRate;
        private int height;

        // private variables
        private IMediaDet mediaDet;
        private double mediaLength;
        private VideoInfoHeader videoInfo;
        private int width;

        /// <summary>
        ///     Creates a FrameGrabber based on the specified video file.
        /// </summary>
        public FrameGrabber(string fileName)
        {
            FileName = fileName;
        }
    }
}
```

```

/// <summary>
    /// Creates a FrameGrabber with no video file set. Set
    /// FileName before calling other methods.
/// </summary>
public FrameGrabber()
    : this("")
{
    // nothing to do here
}

/// <summary>
    /// Gets or sets the full path of the video file from which to
    /// grab frames.
/// </summary>
public string FileName
{
    get
    {
        return fileName;
    }

    set
    {
        mediaDet = null;
        fileName = value;

        if (File.Exists(fileName))
        {
            AMMediaType mediaType = null;

            try
            {
                mediaDet = (IMediaDet)new MediaDet();

                DsError.ThrowExceptionForHR(mediaDet.put_Filename(fileName));

                // find the video stream in the file
                int index = 0;
                Guid type = Guid.Empty;
                while (type != MediaType.Video)
                {
                    mediaDet.put_CurrentStream(index++);
                    mediaDet.get_StreamType(out type);
                }

                // retrieve some measurements from the video
                mediaDet.get_FrameRate(out frameRate);

                mediaType = new AMMediaType();
                mediaDet.get_StreamMediaType(mediaType);
                videoInfo =
                (VideoInfoHeader)Marshal.PtrToStructure(mediaType.formatPtr,
                typeof(VideoInfoHeader));
                DsUtils.FreeAMMediaType(mediaType);
                mediaType = null;
                width = videoInfo.BmiHeader.Width;
                height = videoInfo.BmiHeader.Height;
            }
            catch { }
        }
    }
}

```

```

        mediaDet.get_StreamLength(out mediaLength);
        frameCount = (int)(frameRate * mediaLength);
    }
    catch (Exception e)
    {
        if (mediaType != null)
        {
            DsUtils.FreeAMMediaType(mediaType);
        }

        fileName = "";

        throw new ArgumentException(String.Format("unable to
open the file \"{0}\", DirectShow reported the following error: {1}", value,
e.Message));
    }
}

/// <summary>
/// Gets the total number of frames in the video file.
/// </summary>
public int FrameCount
{
    get
    {
        return frameCount;
    }
}

/// <summary>
/// Gets the framerate of the video file. Some videos always report
"0" regardless of their actual value.
/// </summary>
public double FrameRate
{
    get
    {
        return frameRate;
    }
}

/// <summary>
/// Gets the horizontal dimension of the video file in pixels.
/// </summary>
public int Width
{
    get
    {
        return width;
    }
}

/// <summary>
/// Gets the vertical dimension of the video file in pixels.

```

```

/// </summary>
public int Height
{
    get
    {
        return height;
    }
}

/// <summary>
/// Gets the duration of the video file in seconds.
/// </summary>
public double MediaLength
{
    get
    {
        return mediaLength;
    }
}

/// <summary>
/// Gets the bit depth of the captured frame (always 24 bpp RGB).
/// </summary>
public PixelFormat PixelFormat
{
    get
    {
        return PixelFormat.Format24bppRgb;
    }
}

/// <summary>
/// Gets the image at the specified time. Equivalent to calling
"GetImageAtTime()".
/// </summary>
public Bitmap this[double seconds]
{
    get
    {
        return GetImageAtTime(seconds);
    }
}

/// <summary>
/// Gets the specified frame. Equivalent to calling "GetImage()".
/// </summary>
public Bitmap this[int frame]
{
    get
    {
        return GetImage(frame);
    }
}

/// <summary>
/// Converts the frame number to seconds.
/// </summary>

```

```

public double ConvertFrameNumberToSeconds(int frameNumber)
{
    return (frameNumber / frameRate);
}

/// <summary>
/// Converts the seconds to frame number.
/// </summary>
public int ConvertSecondsToFrameNumber(double seconds)
{
    return (int)Math.Floor(seconds * frameRate);
}

/// <summary>
/// Gets the specified frame. Frames are indexed starting at 0 and
go through (FrameCount - 1).
/// </summary>
public Frame GetFrame(int frameNumber)
{
    return new Frame(GetImage(frameNumber), frameNumber,
ConvertFrameNumberToSeconds(frameNumber));
}

/// <summary>
/// Gets the image of the specified frame. Frames are indexed
starting at 0 and go through (FrameCount - 1).
/// </summary>
public Bitmap GetImage(int frameNumber)
{
    if (frameNumber < frameCount)
    {
        return
GetImageAtTime(ConvertFrameNumberToSeconds(frameNumber));
    }
    else
    {
        throw new ArgumentException(String.Format("frameNumber must
be between 0 and {0} inclusive, value was \"{1}\"", frameCount - 1,
frameNumber));
    }
}

/// <summary>
/// Gets the frame at the specified time. Seconds must be less than
or equal to MediaLength.
/// </summary>
public Frame GetFrameAtTime(double seconds)
{
    return new Frame(GetImageAtTime(seconds),
ConvertSecondsToFrameNumber(seconds), seconds);
}

/// <summary>
/// Gets the image at the specified time. Seconds must be less than
or equal to MediaLength.
/// All other image and frame accessors call this method.

```

6.1.2 Normalize Video

As discussed earlier the video input received at our main interface contains captured frames from a number of cameras binded together as a video stream through an onboard video processor. Since we know the placement of frames from different cameras in the video hence the video can be broken every second to retrieve frames of each camera separately. But for this we need a start point or the first frame to start counting from. For this purpose we have implemented the function for normalization. It captures the first group of six frames that would always contain frames from both main as well as auxiliary camera as per the design of onboard MPEG standard being used. For each image it calculates the RGB color values for each pixel and saves them in a separate data structure. Then it calculates the difference in value for each pixel for each pair of images captured.

This calculated difference of each pixel is then summed up and average value of difference in complete image is calculated. Hence it provides us with the numeric values of difference between each pair of images. This difference is then used to find out whether or not the images belong to same camera. Basing on experimental results we have introduced a threshold value of difference in frames that decides the above. So amongst the subset of six frames we can exactly select the starting image of sequence for our use by discarding anything less than six images in the start.

6.2 Video Display

Video display is required to split the incoming video into respective camera frames and display them separately. The number of cameras which are to be displayed is selected by the viewer. So this requirement is implemented through the PLAYFRAMES () function. This function uses the captured and normalized video and accesses it frame by frame. Keeping in view the known organization of the frames in the video a display algorithm is developed that displays the frames as per their respective cameras at different display panels. Since we know that for every set of 30 frames per second, main camera has 25 frames and aux cameras have one frame each so while displaying these frames we display each aux frame 25 times to keep the frame rate of the video constant. Implementation of this playframes function is given as code underline

```
void playFrames()
{
    FrameGrabber fg = new FrameGrabber(inputPath);
    string outpaths = "C:\\Documents and
                    Settings\\Buraq\\Desktop\\demo frames";
    int counter = 0;
    int p1 = 4, p2 = p1+6, p3 = p2+6, p4 = p3+6, p5 =p4+ 6, p6 =p5+6;
    int x = 36;

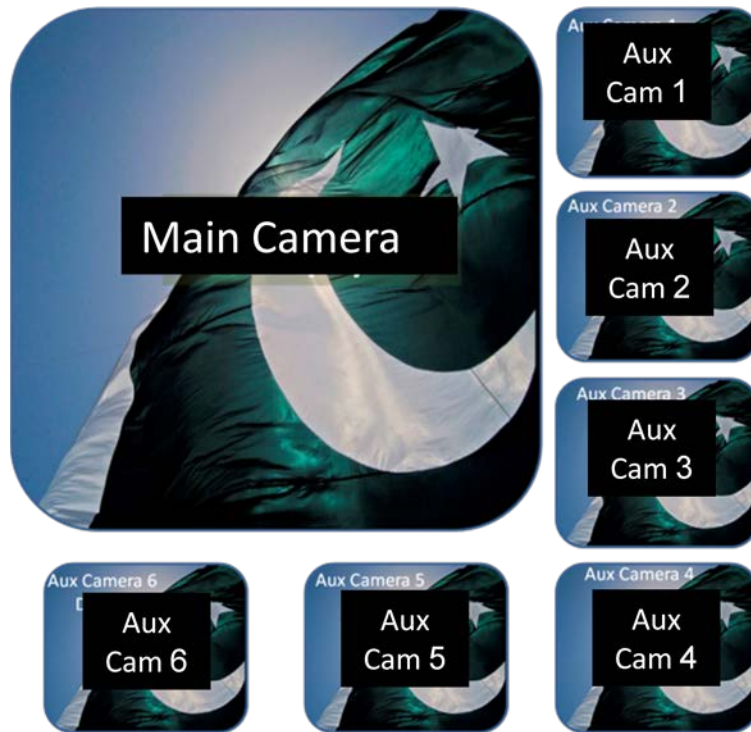
    foreach (FrameGrabber.Frame f in fg)
    {

        counter++;
        if (f.FrameIndex > 2)
        {
            if ((f.FrameIndex-3) % 6 > 0)//to get video after
                removing the aux frames
            {
                mainDisplay.Image = fg.GetImage(f.FrameIndex);
            }
            else
            {
                if (counter == p1 && cstatus[1])
```

```

    {
        p1 += x;
        for (int hh = 0; hh < 25; hh++)//output aux
            frames(25 times each)
            Aux1.Image = fg.GetImage(f.FrameIndex);
    }
    else if (counter == p2 && cstatus[2])
    {
        p2 += x;
        for (int hh = 0; hh < 25; hh++)//output aux
            frames(25 times each)
            Aux2.Image = fg.GetImage(f.FrameIndex);
    }
    else if (counter == p3 && cstatus[3])
    {
        p3 += x;
        for (int hh = 0; hh < 25; hh++)//output aux
            frames(25 times each)
            Aux3.Image = fg.GetImage(f.FrameIndex);
    }
    else if (counter == p4 && cstatus[4])
    {
        p4 += x;
        for (int hh = 0; hh < 25; hh++)//output aux
            frames(5 times each)
            Aux4.Image = fg.GetImage(f.FrameIndex);
    }
    else if (counter == p5 && cstatus[5])
    {
        p5 += x;
        for (int hh = 0; hh < 25; hh++)//output aux
            frames(25 times each)
            Aux5.Image = fg.GetImage(f.FrameIndex);
    }
    else if (counter == p6 && cstatus[6])
    {
        p6 += x;
        for (int hh = 0; hh < 25; hh++)//output aux
            frames(25 times each)
            Aux6.Image = fg.GetImage(f.FrameIndex);
    }
    }
}
Application.DoEvents();
}
}

```

Multi Panel Display

6.3 Video Generator

This component is required to overcome the limitations of the existing MPEG standard being used which generates the video sequence using frames captured by different cameras. It creates instance of our video capture class and then accesses the video at each frame level. Depending upon the user's selection, frames from selected cameras are added separately into video streams. These video streams are created using Microsoft Audio Video Interleave File support library also known as Avifil32.dll. Avifil32.dll is a 32/64-bit Dynamic Linked Library of code components for a

graphics UI style application. This library is used in our customized class Video Stream. CreateStream () function of this class creates separate streams as required for each camera video. Again depending upon the selection algorithm, frames from each camera are separately added to their respective streams using Add Frame() function of our video stream class, hence creating AVI files each containing frames from a single camera. These created AVI files are a sequence of uncompressed images, each comprising of frames captured by a single camera. Here again the mismatch of numbers in frames is covered by adding each frame of aux cameras 25 times. Hence this component generates multiple AVI streams of same length but from different cameras with no inherent flicker due to onboard MPEG standard. The code to generate these separate videos is given as under

```

FrameGrabber fg1 = new FrameGrabber(inputPath);
        //AviManager finalManager = new AviManager("C:\\finaltest1.avi",
false);
        //VideoStream finalStream = finalManager.AddVideoStream(false,
30, fg1.GetImage(0));

        if (inputPath != null && (maincb.Checked || aux1cb.Checked ||
aux2cb.Checked || aux3cb.Checked || aux4cb.Checked || aux5cb.Checked ||
aux6cb.Checked))
        {
            int p1, p2, p3, p4, p5, p6;
            p1 = 3;
            p2 = p1 + 6;
            p3 = p2 + 6;
            p4 = p3 + 6;
            p5 = p4 + 6;
            p6 = p5 + 6;
            //FrameGrabber fg1 = new FrameGrabber(inputPath);
            Bitmap bmp1 = (Bitmap)fg1.GetImage(4);
            Bitmap bmp2 = (Bitmap)fg1.GetImage(p1);
            Bitmap bmp3 = (Bitmap)fg1.GetImage(p2);
            Bitmap bmp4 = (Bitmap)fg1.GetImage(p3);
            Bitmap bmp5 = (Bitmap)fg1.GetImage(p4);
            Bitmap bmp6 = (Bitmap)fg1.GetImage(p5);
            Bitmap bmp7 = (Bitmap)fg1.GetImage(p6);
            AviManager aviManager1 = null;
            AviManager aviManager2 = null;
            AviManager aviManager3 = null;

```

```

    AviManager aviManager4 = null;
    AviManager aviManager5 = null;
    AviManager aviManager6 = null;
    AviManager aviManager7 = null;
    VideoStream aviStream1 = null; ;
    VideoStream aviStream2 = null;
    VideoStream aviStream3 = null;
    VideoStream aviStream4 = null;
    VideoStream aviStream5 = null;
    VideoStream aviStream6 = null;
    VideoStream aviStream7 = null;
    // if (maincb.Checked)
    //{
        aviManager1 = new AviManager("C:\\Documents and
Settings\\Buraq\\Desktop\\demo\\main.avi", false);
        aviStream1 = aviManager1.AddVideoStream(false, 25,
bmp1);
    // }
    //if (aux1cb.Checked)
    //{
        aviManager2 = new AviManager("C:\\Documents and
Settings\\Buraq\\Desktop\\demo\\Aux1.avi", false);
        aviStream2 = aviManager2.AddVideoStream(false, 25,
bmp2);
    // }
    //if (aux2cb.Checked)
    //{
        aviManager3 = new AviManager("C:\\Documents and
Settings\\Buraq\\Desktop\\demo\\Aux2.avi", false);
        aviStream3 = aviManager3.AddVideoStream(false, 25,
bmp3);
    //}
    //if (aux3cb.Checked)
    //{
        aviManager4 = new AviManager("C:\\Documents and
Settings\\Buraq\\Desktop\\demo\\Aux3.avi", false);
        aviStream4 = aviManager4.AddVideoStream(false, 25,
bmp4);
    //}
    //if (aux4cb.Checked)
    //{
        aviManager5 = new AviManager("C:\\Documents and
Settings\\Buraq\\Desktop\\demo\\Aux4.avi", false);
        aviStream5 = aviManager5.AddVideoStream(false, 25,
bmp5);
    //}
    //if (aux5cb.Checked)
    //{
        aviManager6 = new AviManager("C:\\Documents and
Settings\\Buraq\\Desktop\\demo\\Aux5.avi", false);
        aviStream6 = aviManager6.AddVideoStream(false, 25,
bmp6);
    //}
    //if (aux6cb.Checked)
    //{
        aviManager7 = new AviManager("C:\\Documents and
Settings\\Buraq\\Desktop\\demo\\Aux6.avi", false);

```

```

        aviStream7 = aviManager7.AddVideoStream(false, 25,
bmp7);
        // }
        Bitmap b1, b2, b3, b5, b6, b7; ;
        int counter = 2;

for(int i=3 ;i<fg1.FrameCount;i++)
    {
        counter++;
        if ((i - 3 ) % 6 > 0) && maincb.Checked)
        {
            //p2 += 36;
            // for (int hh = 0; hh < 25; hh++)//output aux
frames(5 times each)
            //{
                b1 = (Bitmap)fg1.GetImage(i);
                aviStream1.AddFrame(b1);
                b1.Dispose();
            //}
        }
        else if (counter == p1 && aux1cb.Checked)
        {
            p1 += 36;
            for (int hh = 0; hh < 25; hh++)//output aux frames(5
times each)
                {
                    b2 = (Bitmap)fg1.GetImage(i);
                    aviStream2.AddFrame(b2);
                    b2.Dispose();
                }
        }
        else if (counter == p2 && aux2cb.Checked)
        {
            p2 += 36;
            for (int hh = 0; hh < 25; hh++)//output aux frames(5
times each)
                {
                    b3 = (Bitmap)fg1.GetImage(i);
                    aviStream3.AddFrame(b3);
                    b3.Dispose();
                }
        }
        else if (counter == p3 && aux3cb.Checked)
        {
            p3 += 36;
            for (int hh = 0; hh < 25; hh++)//output aux frames(5
times each)
                {
                    //b4 = (Bitmap)fg1.GetImage(i);
                    aviStream4.AddFrame((Bitmap)fg1.GetImage(i));
                    // b4.Dispose();
                }
        }
        else if (counter == p4 && aux4cb.Checked)

```

```

    {
        p4 += 36;
        for (int hh = 0; hh < 25; hh++)//output aux frames(5
times each)
        {
            b5 = (Bitmap)fg1.GetImage(i);
            aviStream5.AddFrame(b5);
            b5.Dispose();
        }
    }
    else if (counter == p5 && aux5cb.Checked)
    {
        p5 += 36;
        for (int hh = 0; hh < 25; hh++)//output aux frames(5
times each)
        {
            b6 = (Bitmap)fg1.GetImage(i);
            aviStream6.AddFrame(b6);
            b6.Dispose();
        }
    }
    else if (counter == p6 && aux6cb.Checked)
    {
        p6 += 36;
        for (int hh = 0; hh < 25; hh++)//output aux frames(5
times each)
        {
            b7 = (Bitmap)fg1.GetImage(i);
            aviStream7.AddFrame(b7);
            b7.Dispose();
        }
    }
}
aviManager1.Close();
aviManager2.Close();
aviManager3.Close();
aviManager4.Close();
aviManager5.Close();
aviManager6.Close();
aviManager7.Close();

```

6.4 Video Compressor

A component required to compress the created videos by video generator. It takes input the generated AVI files and compresses them into compressed video streams. This component is also implemented using windows API Avifil32.dll. This API is used to

create our new class of AVI file that implements functions to create compressed streams.

Since the onboard MPEG standard generates the compressed stream of frames captured by different onboard cameras hence it has an inherent overhead of keeping more key frames. For every 30 frames in a second, first five frames of main camera are compressed and then a frame is received from one of the aux camera hence is saved as key frame, next packet of five frames would again require an additional key frame to be saved. Therefore the existing MPEG standard has an overhead of saving 11 additional key frames every second and also a disturbing flicker due to placement of aux frames between packets of main frames. Our algorithm however overcomes both these problems, firstly by removing the overhead of 11 additional key frames, as all frames are from same camera so every second requires only one key frame and secondly by removing the flicker as each generated compressed video stream comprises of frames from a single selected camera.

6.5 Audio Data Reader

This component is implemented using the second approach discussed above in design where we are reading the data hidden in audio through the existing media player. The memory distribution and location of data required is found through detailed analysis and study of the existing system using disassemblers and Decompilers. This process involved detailed study of around fifteen thousand lines of assembly code generated through disassembling the sourceless existing media player. The study was carried out

using a number of tools available for memory tracing and tracking, generating assemblies from machine code of an executing application. The distribution of memory and virtual addressing of existing media player was studied, locations and addresses of the desired data was found and traced, and using results of this study algorithm to retrieve the data was formulated.

The existing player is embedded into our application and during its execution the respective memory locations are accessed to get the desired data. This access of memory is carried out using a dynamic code injection technique. In dynamic code injection a predefined instruction set is injected into an ongoing process for execution. Using a code snippet for reading process memory the existing media player is executed and desired information is retrieved and written to a desired location. Implementation is shown through the following code snippet

```
namespace Burraq
{
    class ProcessMemoryReaderApi
    {
        public const uint PROCESS_VM_READ = (0x0010);

        [DllImport("kernel32.dll")]
        public static extern IntPtr OpenProcess(UInt32 dwDesiredAccess, Int32
bInheritHandle, UInt32 dwProcessId);

        [DllImport("kernel32.dll")]
        public static extern Int32 CloseHandle(IntPtr hObject);

        [DllImport("kernel32.dll")]
        public static extern Int32 ReadProcessMemory(IntPtr hProcess, IntPtr
lpBaseAddress, [In, Out] byte[] buffer, UInt32 size, out IntPtr
lpNumberOfBytesRead);
    }
    //-----
    public class ProcessMemoryReader
    {
        public ProcessMemoryReader()
        {
```

```

    }

    public Process ReadProcess
    {
        get
        {
            return m_ReadProcess;
        }
        set
        {
            m_ReadProcess = value;
        }
    }

    private Process m_ReadProcess = null;

    private IntPtr m_hProcess = IntPtr.Zero;

    public void OpenProcess()
    {
        m_hProcess =
ProcessMemoryReaderApi.OpenProcess(ProcessMemoryReaderApi.PROCESS_VM_READ, 1,
(uint)m_ReadProcess.Id);
    }

    public void CloseHandle()
    {
        int iRetValue;
        iRetValue = ProcessMemoryReaderApi.CloseHandle(m_hProcess);
        if (iRetValue == 0)
            throw new Exception("CloseHandle failed");
    }

    public byte[] ReadProcessMemory(IntPtr MemoryAddress, uint
bytesToRead, out int bytesReaded)
    {
        byte[] buffer = new byte[bytesToRead];

        IntPtr ptrBytesReaded;
        ProcessMemoryReaderApi.ReadProcessMemory(m_hProcess,
MemoryAddress, buffer, bytesToRead, out ptrBytesReaded);

        bytesReaded = ptrBytesReaded.ToInt32();

        return buffer;
    }
}
}
}

```


6.6 Frame Tagger

A component that is required to save a selected frame tagged with its geographical data at a desired location. It is also implemented through our video capture functions. Image of any location in video can be retrieved from display and written to a file through this component.

Chapter 7

Results and Analysis

6.1 [Mpeg standard](#)

Existing Result

It provides a video with inherent flicker due to customized implementation of MPEG.

Achieved Result

The new implementation of video generating algorithm and MPEG standard removes the flicker and generates smooth video.

7.2 [Video display](#)

Existing Result

It has a single display panel that shows the video from all cameras in the same stream.

Achieved Result

Now we have separate display of panel for each camera (MAIN AND AUXILLARIES) showing a smooth and flicker less video.

7.3 [Audio data reader](#)

Existing Result

It provides geographical data in read only format preventing any further use of the data.

Achieved Result

It reads encrypted data in the form that can not only be saved but can be further used for automated applications like goggle.

7.4 Compression

Existing Result

At present compression algorithm comes with an inherent overhead of saving additional key frames for every second of video.

Achieved Result

Now we have overcome this additional overhead by generating separate videos frames captured by each camera.

7.5 Video Generator

Existing Result

It creates a single stream of video comprising of frame captured by different cameras hence generating an interlaced video.

Achieved Result

Now it generates separate video streams comprising of frames from separate cameras thus removing interlacing limitation.

7.6 Frame Extractor

Existing Result

It provides no means for extracting individual frames from the video thus making it a virtually dumb module.

Achieved Result

Now it can retrieve and save any frame from the video at display.

Chapter 7

Conclusions and Future

Work

Conclusion and Future Work

In this chapter we compare the already existing system with the one we have made. The existing system of BURRAQ has some constraints which limits the working of the system and also its usage in other applications. We have tried to improve the existing system so that it can work better to improve the efficiency of the system and also it can be used in future applications as well.

As already discussed, the existing system has some limitations which hindered its smooth and efficient working. Currently the deployed system has the following limitations

- a) The audio signal containing the Geographical data of the underlying video is readable only to the 'UAV Media Player' and remains hidden for any other module to work with.
- b) The GUI of the Ground Control System provides the geographical data only in 'read only mode' preventing any further use of the data.
- c) Existing GUI is virtually dumb providing no usable output for further use in area mapping and marking.
- d) The MPEG standard provides a flickering video making it almost useless for any tactical analysis and planning.

We have made an endeavor to make the enhanced existing system which work better and have tried to eradicate the beyond discussed limitations. We have also integrated new functionalities in the system, which make our system superior with the

existing version in many aspects. Our improved system has the following new functionalities as follow scan be used in other applications as well.

- a) A GUI is provided to the existing module with an aim of providing the readable data in usable form for further mapping applications.
- b) A new MPEG standard is developed to reduce/remove the flickering effects in the video stream hence providing a smooth video stream for tactical and planning analysis.
- c) The developed interface can be utilized to use the Geographical data to map the video over digital mapping applications like Google Maps.
- d) The extracted data can be used to create maps of specified areas for further tactical/strategic use.

The above discussed implementations will improve the efficiency of the existing system in future applications.

Chapter 8

Bibliography

1. http://www.sersc.org/journals/IJDTA/vol2_no2/1.pdf
2. <http://www.mpeg.org>
3. <http://en.wikipedia.org/wiki/Disassembler>
4. http://en.wikipedia.org/wiki/Static_code_analysis
5. <http://www.coverity.com/products/dynamic-analysis.html>
6. <http://en.wikipedia.org/wiki/Decompiler>
7. <http://www.afterdawn.com/glossary/term.cfm/mpeg-4>
8. <http://www.webopedia.com/TERM/M/MPEG.html>
9. <http://cs.utsa.edu/~jortiz/CS4953/Papers/Techniques%20for%20Data%20Hiding-p.pdf>
10. <http://en.wikipedia.org/wiki/MPEG-4>
11. http://en.wikipedia.org/wiki/Optical_character_recognition
12. <https://buildsecurityin.us-cert.gov/bsi/214-BSI.html>
13. http://en.wikipedia.org/wiki/Dynamic_program_analysis
14. <http://www.videoanalysis.org/>
15. <http://www.springerlink.com/index/k662l007375m1823.pdf>
16. <http://mpeg.chiariglione.org/standards/mpeg-4/mpeg-4.htm>
17. <http://www.mpeg4.net/>
18. http://images.apple.com/quicktime/pdf/MPEG4_v3.pdf
19. <http://www.see.ed.ac.uk/~mjj/dspDemos/EE4/index.html>
20. http://www.computerworld.com/s/article/73023/Optical_Character_Recognition
21. <http://www.metacarta.com/Collateral/Documents/English-US/OCR-Kornai.pdf>

22. [http://www.scar.ac.cn/hhkxen/ch/reader/create_pdf.aspx?file_no=2010302&flag=1
&journal_id=hhkxen](http://www.scar.ac.cn/hhkxen/ch/reader/create_pdf.aspx?file_no=2010302&flag=1&journal_id=hhkxen)
23. <http://www.hydrogenaudio.org/forums/lofiversion/index.php/t4292.html>
24. http://www.sersc.org/journals/IJDTA/vol2_no2/1.pdf